Draft 20 Feb 97

Evaluation of Assembly Simulators Used in Closed Loop Attitude Control System Testing

by

Jason Christopher Bunn

S.B. Aeronautics and Astronautics
Massachusetts Institute of Technology, 1996

Submitted to the Department of Aeronautics and Astronautics

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN AERONAUTICS AND ASTRONAUTICS

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June, 1997

Signature of Author: ______________________________
Department of Aeronautics and Astronautics
XX, XX,1997

Certified by: ______________________________
Steven R. Hall
Associate Professor of Aeronautics and Astronautics
Thesis Supervisor

Approved by: ______________________________
Professor Jaime Peraire
Associate Professor of Aeronautics and Astronautics
Chairman, Departmental Graduate Office

Evaluation of Assembly Simulators Used in Closed Loop Attitude Control System Testing

by

Jason Christopher Bunn

Submitted to the Department of Aeronautics and Astronautics

on February 20,1997 in Partial Fulfillment of the Requirements

for the Degree of Master of Science in Aeronautics and Astronautics

ABSTRACT

The Cassini spacecraft's Attitude and Articulation Control Subsystem has been tested extensively at the Jet Propulsion Laboratory in Pasadena, California. Three of the subsystem's assemblies have been tested using assembly simulators in place of actual hardware. These simulators have been designed and tested to ensure as much commonality with the hardware as possible. Several early design choice have impacted the degree to which the simulators have matched the hardware, the most crucial decision concerning the interface between the assembly simulators and the flight hardware. However, these difficulties were overcome and all testing requirements were satisfied.

The use of simulators has resulted in increased testing ability due to the small number of flight components constructed. Though this experience, several key lessons were learned, chief among them being clear definition of expectations and the importance of defining simulation interfaces as identical as possible to the flight equipment. Assembly simulators, properly developed, should prove a valuable alternative to physical hardware testing for future flight projects.

Thesis Advisor: Professor Steven R. Hall
Associate Professor of Aeronautics and Astronautics

Acknowledgments

First, I would like to thank the Engineering Internship Program office at MIT for making the opportunity to conduct research at the Jet Propulsion Laboratory possible. Iwouldalso liketo thank the Co-Op office at the Jet Propulsion Laboratory for giving me the opportunity to participate. For their support and assistance, I would like to thank the Validation Group of the Avionic Systems Engineering Section at JPL, especially Richard Haga, Mario Mora, James Roberts, as well as Roy Okuno of Boeing and Andrew Engelmann of the University of Colorado. Finally, I wish to thank my parents, Anthony and Patricia Bunn, for their constant support and love over the years.

Acronym List

AACS	Attitude and Articulation Control Subsystem
ACC	A c c e l e r o m e t e r
AFC	AACS Flight Computer
ALF	Accelerated Load Format
ATLO	Assembly, Test and Launch Operations
BAIL	Backdoor ALF Injection Loader
BPLVD	BiPropellant Latch Valve Driver
CATS	Cassini AACS Test Station
CDS	Command and Data Subsystem
DARTS	Dynamics Algorithms for Real Time Simulation
EFC	Engineering Flight Computer
EGA	Engine Gimbal Actuator
EGECU	Engine Gimbal Electronics Control Unit
EGED	Engine Gimbal Electronics Driver
GMT	Greenwich Mean Time
FSDS	Flight Software Development System
HELVD	Helium Latch Valve Driver
IRU	Inertial Reference Unit
lTL	Integration and Test Laboratory
JPL	Jet Propulsion Laboratory
MEVD	Main Engine Valve Driver
MPD	MonoPropellant Driver
Pcu	Power Conversion Unit
PIU	Pixel Input Unit
PMS	Propulsion Module Subsystem
Pou	Pixel Output Unit
PPS	Power and Pyrotechnic Subsystem
RWA	Reaction Wheel Assembly

RWM	Reaction Wheel Motor
RWX	Reaction Wheel Electronics
SEHW	Support Equipment Hardware
SESW	Support Equipment Software
SSH	Sun Sensor Head
SSE	Sun Sensor Electronics
SSPS	Solid State Power Switch
SRU	Stellar Reference Unit
VDECU	Valve Drive Electronics Control Unit
VDEX	Valve Drive Electronics Extension

Chapter 1

Introduction

In October 1997, a Titan IV/Centaur launch vehicle will lift the Cassini spacecraft towards Saturn, beginning the last in a series of grand tour missions that included the Voyager probes and the Galileo spacecraft. Cassini's mission is to deliver the Huygens Titan probe to the surface of Titan and to perform an orbital analysis of the Saturinan system. The spacecraft has been developed and tested primarily by the California institute of Technology's Jet Propulsion Laboratory (JPL) for NASA's Office of Space Science.

The attitude and articulation control subsystem (AACS) for Cassini has been extensively tested at JPL through a comprehensive closed loop testing plan. The AACS has a number of inputs to allow for support equipment to simulate the outside environment of the spacecraft during the mission. Examples of these inputs include accelerometer and gyroscope biases, simulated starfields and simulated sun sensor inputs. Interfaces external to the AACS are also simulated. These include the Command and Data Subsystem (CDS), the Power and Pyrotechnic Subsystem (PPS) and the Propulsion Module Subsystem (PMS). This allows for the hardware to experience flightlike conditions (with the exception of environmental conditions) and makes closed loop testing possible.

However, in many cases it is not possible for a complete set of AACS hardware to undergo this extensive testing. During the busiest testing period, three laboratories are running simultaneously at JPL for AACS testing. For most of the hardware, there exists sufficient flight hardware to accommodate these laboratories. But for three assemblies, this is not the case. Due to the high cost of these assemblies and the quicker development time of simulators, the reaction wheel assembly, the inertial reference unit and the engine gimbal actuators are replaced by assembly simulators during closed loop testing.

These simulators play a pivotal role in AACS testing. During pre flight testing, the simulators must act sufficiently like the flight units to permit testing of software functionality and mission sequences. After launch, these simulators are even more important as they become the only mechanism by which testing of sequences or anomalies can occur.

This thesis evaluates the effectiveness of these simulators during testing of the AACS at JPL. With all of the laboratories relying on these simulators after launch, it is very important to understand the consequences of this approach. As the trend to drive down develop costs in space missions continues, there is every reason to expect that future missions will have to rely on assembly simulators in an ever increasing capacity. Evaluation of these assemblies of the Cassini spacecraft is a step toward developing a knowledge base that could be used when considering different testing options in the future.

This thesis begins with a description of the closed loop testbed at JPL and how the laboratory environment works. This is followed by an analysis of the testing of the engine gimbal actuators, the inertial reference unit and the reaction wheel assembly. Finally, we conclude with an evaluation of the testbed as a whole as well as lessons learned during closed loop simulation.

Chapter 2

Closed Loop Attitude Control Testing

2.1 AACS

The attitude and articulation control subsystem is responsible for attitude determination and control during all phases of the mission. The components of the AACS are shown in figure 2.1 and are described briefly below.

- AACS Flight Computer (AFC): The AFC is responsible for acting on commands from the command and data subsystem (CDS) concerning guidance, navigation and control. The system is dually redundant and interfaces with the AACS databus, the CDS databus and the power and pyrotechnic subsystem (PPS). The AFC direct access port is used to send commands to the AFC when the CDS is not present during test. The AFC also interfaces with the Stellar Reference Unit through a dedicated databus used to gather pixel information during Star Identification.
- Accelerometer (ACC): The accelerometer is used to determine changes in velocity along the spacecraft Z axis. The accelerometer is a non redundant component that interfaces with the PPS and the AACS databus. The accelerometer has a direct access port that allows for simulation of Z axis acceleration during testing.
- Backdoor ALF Injection Loader (BAIL): This assembly is a fault protection device that is used to provide a level of redundancy in the event that CDS has difficulty loading the AFC with its soft ware. The BAIL contains accelerated load format (ALF) data blocks that can load the AFC in the event of problems with nominal loading via CDS.
- Engine Gimbal Electronics/Actuators (EGE/EGA): The EGA's articulate the gimbals attached to the dual main engines of the spacecraft. The EGA's are controlled by the EGEs, which interface with the AACS databus.
- Inertial Reference Unit (IRU): The dual redundant IRU uses a set of four hemispherical resonating gyros (HRG) to provide inertial rate information to the AFC for use in attitude control.
- Reaction Wheel Assembly (RWA): There are four RWA's on the spacecraft. Three of these are arranged for use as actuators during attitude control. The fourth wheel is redundant and can be positioned to replace any of the three primary wheels in the event of a failure.
- Stellar Reference Unit (SRU): The dual redundant SRU is a sensor used to determine position

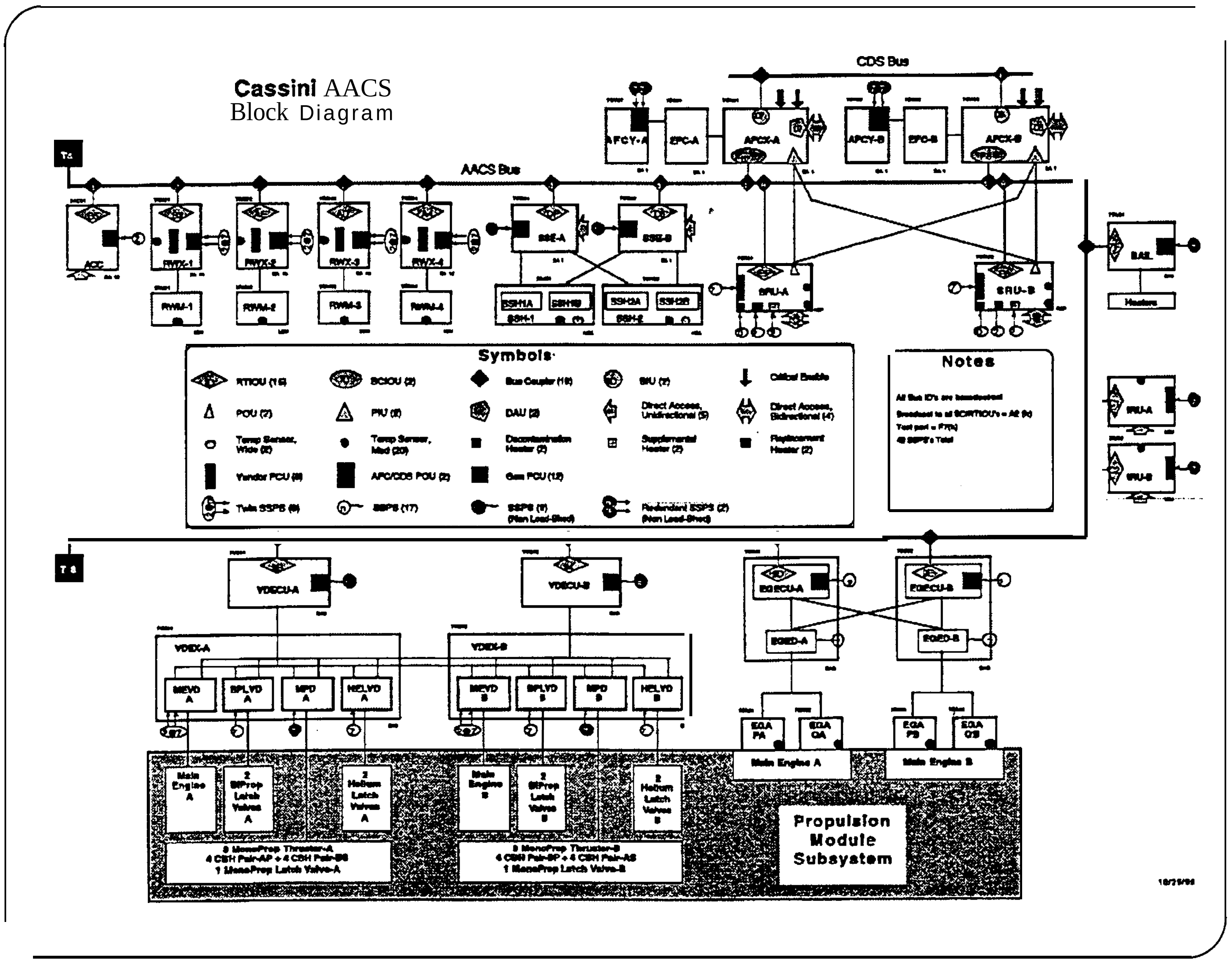


Figure 2-1 Attitude and Articulation Control Subsystem Block Diagram

and attitude during the cruise portion of Cassini's mission. It utilizes a charged coupled device (CCD) camera to detect bright bodies in its field of view and this data is passed to the AFC via a pixel interface unit (PIU) for processing.

- Sun Sensor Assembly (SSA): The dual redundant sun sensor assembly is used to detect the position of the sun during attitude determination. Sun sensor heads are placed on the high gain antenna and generate voltages proportional to the amount of light that hits the heads.
- Valve Drive Electronics (VDE): The valve drive electronics are used to interface with the Propulsion Module Subsystem to open and close the various thrusters used for attitude control and as well as the main engine valves.

2.2 AACS Closed Loop Testbeds

2.2.1 ITL

The primary lab for integration and test of the AACS at the subsystem level is the Integration and Test Laboratory (ITL). The purpose of the ITL is to perform hardware integrations of the AACS subsystem and test functional sequences of the Cassini mission. All flight hardware as well as engineering models or flight spares are tested in the ITL to ensure proper electrical configuration when the assembly is integrated with the AACS databus, the Power and Pyrotechnic Subsystem (PPS) and the Propulsion Module Subsystem, if applicable (VDE, EGE/EGA). For this purpose, there are hardware simulators of the PPS and PMS in the ITL that allow testing of the interfaces to these subsystems.

For testing functional sequences, the ITL has an extensive set of support equipment hardware and software. The hardware and software work together to simulate the external interfaces to the AACS and permits closed loop testing. Support equipment hardware consists of a series of computers and additional equipment which interface with the users of the system as well as the hardware. This equipment includes an Inertial Sensors Controller (INS Controller) that permits biasing of the accelerometer and inertial reference unit, the assembly simulator hardware for the engine gimbal actuators, the inertial reference unit and the reaction wheel assembly, electronics to generate bias voltages to send to the sun sensor assembly and star field data to send to the stellar reference unit, and equipment to interface with the AACS Flight Computer. The hardware also includes the simulators for the command and data subsystem, the PPS and the PMS.[2]

Support equipment software is an extensive network of computer programs working to simulate the outside environment of the spacecraft. The programs fall into two large groups-- real time and non real time. Real time programs consist of the assembly simulator software, subsystem simulation soft ware (e.g. CDS, PPS, PMS), as well as software to monitor different assemblies and report on their status. The non-real time software includes tasks such as user console interfacing and generation of displays.

The software also uses the concept of a “blackboard” to provide for data visibility across the simulation. The software runs on a set of five processors called chassis. These processors must work in a synchronized fashion and timing is very critical. Therefore, the processors use shared memory to facilitate data transfer. All of the chassis use the same set of memory to read and write variables. This means that all of the processors knows what the state of the system is at any given time. This is analogous to how a common blackboard is used so that all those in a room have a consistent data set, and thus this reflective memory is know as the blackboard. [4]

Finally, the dynamics of the simulation are propagated in real time via a computer program called DARTS- Dynamic Algorithms for Real Time Simulation. This program was developed by A. Jain of the Jet Propulsion Laboratory and computes the time rate of change of the state of the Cassini spacecraft as a dynamical system[3]. The program accepts any number of actuators, sensors and flexible modes and thus is what makes closed loop dynamics testing possible for Cassini.

The ITL also permits some level of system mode testing. The Command and Data Subsystem is usually simulated in the ITL, but for some testing, the actual CDS is used and the AACS Flight computer takes its commands from the actual command computer. This permits testing of the CDS-AACS interface in the ITL before assembly, test and launch operations begin.

2.2.2 **Cassini AACS Test Station**

There also exists the Cassini AACS Test Station (CATS) for development of flight and support software. CATS is similar to the ITL in that actual hardware is present in most cases, support equipment hardware processors are used to interface with the hardware and users, and a closed loop dynamics environment is possible with the use of DARTS. The major difference between CATS and ITL is their purpose. ITL is used to test electrical interfaces of the flight hardware and thus all hardware that is flight certified in first tested in the ITL before delivery to the Spacecraft Assembly Facility. CATS, on the other hand, is used to test the software of the system. The hardware is present, but electrical “breadboards” are primarily used in CATS. The functionality of these breadboards is identical to the flight units in most cases, but shielding, grounding and electrical interfaces may be different.

2.2.3 Assembly, Test and Launch Operations

Assembly, Test and Launch Operations (ATLO) for Cassini began in late 1995 and will continue through launch of Cassini in October, 1997. Primarily testing occurs at the Spacecraft Assembly Facility at JPL. At this location, the flight hardware is integrated together for the final time. Also, many of the simulators are not present. The PPS hardware is there, and thus all power comes from the actual PPS. There are no assembly simulators once all the hardware is integrated. The CDS is also present and there is no CDS simulator. However, there is one simulator still present, The PMS simulation is still running in ATLO due to the danger to personnel of testing main engine firings and reaction control thrusters. The Propulsion Module is tested separately by Lockheed Martin for the vast majority of the testing period. During most testing in ATLO, the DARTS simulation is still present and permits closed loop testing. The exception is during environmental testing when the support equipment is disconnected and no closed loop testing is conducted.

Throughout the testing plan, the EGAs, RWAS and IRUs are simulated in the ITL and in CATS. The real hardware is present for interface checkout in the ITL and during ATLO testing. The next sections go into detail concerning the three assembly simulators and evaluates their usage in the testing of Cassini’s AACS.

Chapter 3

Engine Gimbal Actuators

3.1 Description

The purpose of the engine gimbal actuators is to rotate the main engines of the propulsion module subsystem about their gimbal axes in response to commands by the AACS flight computer. The main engines are rotated such that the thrust vector-passes through the center of mass of the space-craft as well as in the desired inertial direction. The IRU and ACC are used in conjunction with the EGA's to determine this direction as well as to determine when the required velocity change has been achieved.

A signal flow for the EGA's is shown in figure 3.1. The actuators act on extension commands pro-

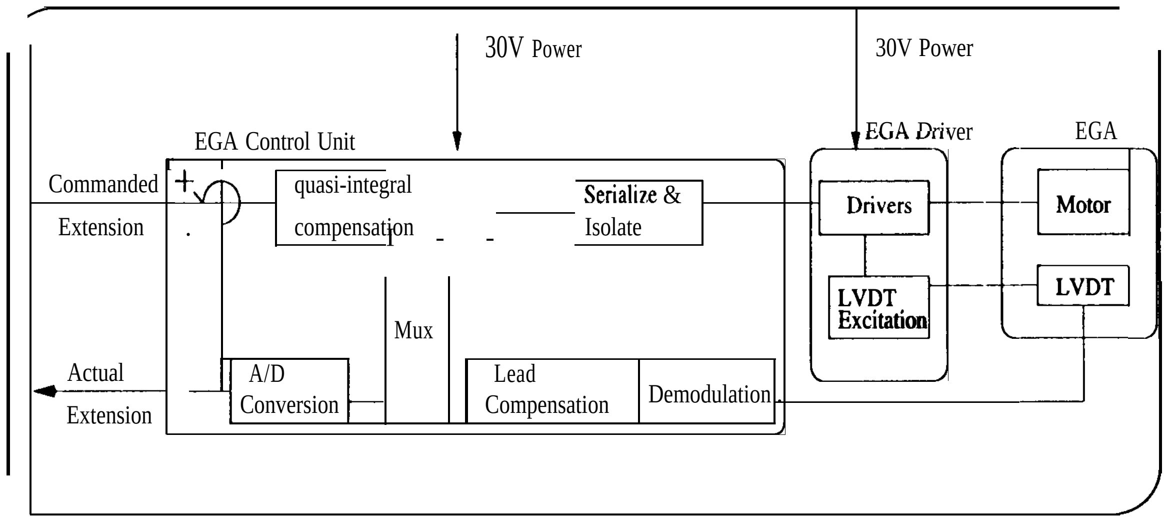


Figure 3-1 Engine Gimbal Actuator Block Diagram

vialed by the AACS flight computer through the engine gimbal electronics. The extension is con-trolled by comparing the actual positions of the actuators to the desired position and correcting the position by altering the motor voltage. The actual positions are determined by the feedback signal from a Linear Variable Differential Transformer (LVDT) that is attached to each actuator. The electronics that control the position consist of a control unit and a driver. The driver accepts volt-age commands from the control unit and generates excitation signals for the motor and the LVDT of the actuator. The control unit accepts commanded extensions off the AACS databus and the

LVDT feedback signal. The control unit generates a digital representation of the LVDT signal for the databus as well as the voltage commands for the driver.

3.2 Cassini Laboratory Configuration

The Engine Gimbal Actuators are simulated in the ITL and CATS through a combination of a hardware simulator and a software dynamics interface. The purpose of the hardware is to simulate the engine gimbal actuators and generate a feedback signal that represents the LVDT signal for the real actuators. The dynamics software accepts the commanded extensions and the LVDT extension information and then computes the main engine thrust vector direction for use in the DARTS simulation.

3.2.1 EGA Hardware Simulator

Description

The EGE hardware simulator was designed to simulate the LVDT feedback information that is supplied by the real EGAs in response to the LVDT excitation signal and the motor drive input. A block diagram is shown in figure 3.2.

To simulate the LVDT, the EGA hardware simulator consists of a motor simulator and the LVDT simulator. The motor simulator accepts the EGE drive signal and passes it through an optical isolator. The signal is scaled such that a position signal is generated that is proportional to the commanded position of the actuator. Then, this signal is multiplied by the LVDT excitation signal, also received from the EGE. The multiplication retains the sign of the motor drive signal and the result thus simulates the LVDT feedback signal.

Validation Test

When the engine gimbal actuators are integrated into the subsystem in the ITL, they are integrated per a hardware integration procedure. This procedure exercises the actuators and ensures that they perform adequately. Since the simulators and the real actuators are integrated using the same procedure, the data can be compared to evaluate the simulators.

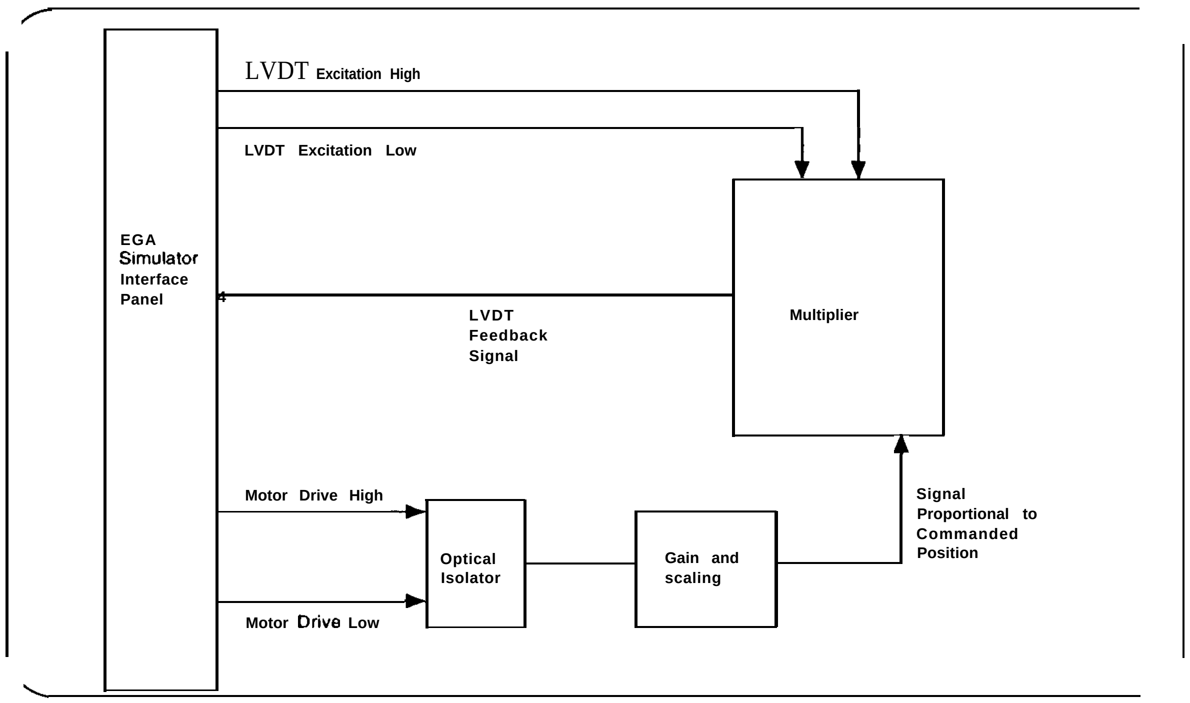


Figure 3-2 Engine Gimbal Actuator Hardware Simulator

The actuators (or simulators) are exercised through a series of extensions during the integration procedure. The results of this procedure was that the simulators perform to within specified tolerances and track the hardware very well and these results are summarized below:

- Average absolute deviation from the commanded position for simulation serial number 005 was 0.0326 mm.
- Average absolute deviation from the commanded position for simulation serial number 010 is was 0.0215 mm.
- Cassini AACS requirement: O. 1 mm deviation.

Even though the AACS requirement was met on average, the maximum error did deviate from the AACS requirement for both of the simulators. This deviation was deemed acceptable for testing, however. This is because Cassini fault protection will activate if the deviation is greater than 0.27 mm for two consecutive readings of the EGA (reading occur every 125 ins). This behavior was not observed when testing either the EGA's or the EGA simulators and thus the EGA simulators were accepted for testing. Due to the variance of the EGA positions, the power measurements of the EGAs also had more variance than with the flight equipment, but this was also acceptable. [5]

Differences and Problems

One problem that did occur during testing is that a fault protection error in flight software was not found in the ITL testing and the bug was discovered when the fault protection autonomously powered down the EGA driver during integration of the flight Propulsion Module Subsystem on the actual spacecraft. The investigation concluded that the EGA simulators were not designed to simulate an EGA under actual flight loading conditions. Therefore, the first time the flight software interfaced with an EGA loaded onto a gimbal and a Main Engine was during ATLO testing. This brought out one of the important lessons learned through AACS testing. If the requirements are not stated clearly at the outset and thought through in their entirety, unforeseen events may occur. The result of this testing in ALTO was a decision to use flight spare EGAs attached to a load fixture for future testing. Thus the EGA simulators will not be used for post launch ITL analysis activities.

3.2.2 **EGA Dynamics Model**

Description

To properly represent the motion of the EGA's in the closed loop dynamics simulation, the support equipment software must ensure that the main engine thrust vector direction is consistent with the EGA extensions. To accomplish this, a software "model" of the EGA accepts information form the EGA (or EGA simulator) and computes the main engine thrust vector. The relationship between the EGA (or simulator) and the EGA "model" is shown in figure **3.3** and is described below.

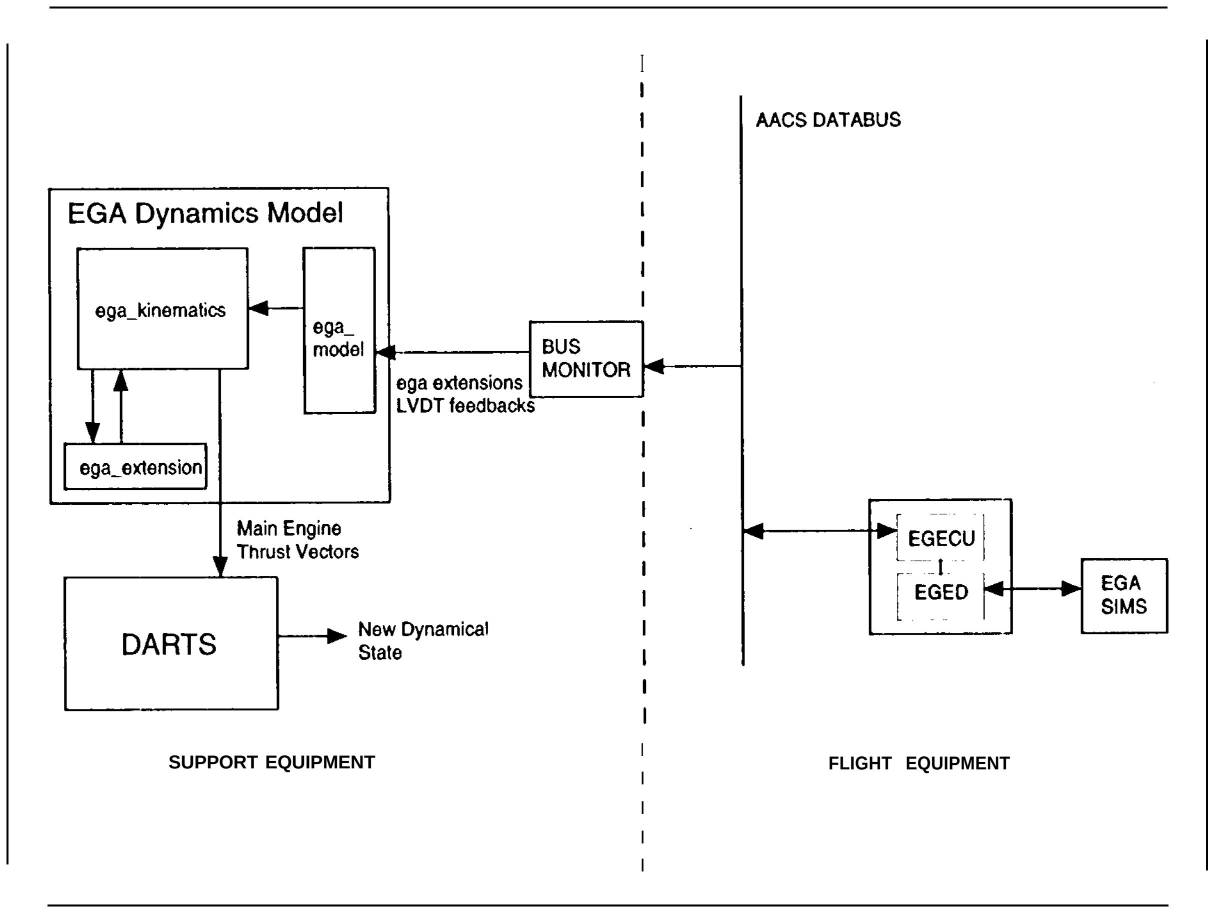


Figure 3-3 Engine Gimbal Actuator Dynamics Model

The model consists of three modules: ega_model, ega_kinematics and ega_extension. The first module, ega_model, accepts the LVDT and commanded extension information and computes the new extensions of the actuators. This is done without regard to dynamics and the extension is simply set to the current commanded extension. The only exception is if the step required to update the position is too high. In this case, the model updates the extension with a series of smaller steps. Once the extensions are computed, ega_kinematics is called. This module calculates the thrust vector by first computing the gimbal angles (accomplished by the module ega_extension) and then transforming the angle information into the main engine thrust vector direction. Once the thrust vector direction is known, this information is used during the next iteration of the dynamics simulation.

Validation Testing and Results

The EGA model was validated by comparing resulting engine gimbal angles to analytical predicts. The EGAs were commanded to several different positions and the computed main engine thrust vector was recorded. Given this thrust vector, a solution for the EGA position was derived and compared to the commanded position. The data was consistent and the model was validated in this isolated case. The table below shows the predicted and actual values for the EGA extensions in response to several command to stroke the gimbal actuators.

'E-GA P Commanded	EGA P Actual	EGA Q Commanded	EGA Q Actual
o	0	19	18
1563	1563	0	0
1563	1563	1563	1563
-155	-155	1563	1564
-155	-155	961	962
-963	-963	961	962
-963	-963	-1563	-1562
155	155	-1563	-1562
155	155	-961	-960
0	0	-961	-960
0	0	-1	0
960	962	-1	1
960	962	-960	-961
-1563	-1562	-960	-959
-1563	-1563	-1563	-1562
-962	-962	-1563	-1562
1563	1563	961	961
1563	1563	1563	1563
0	0	1563	1564
0	0	0	1

3.2.3 **Closed Loop Simulation**

The two primary test activities of the laboratory that have validated the described EGA simulation have been the Main Engine Trajectory Correction Maneuver (TCM) Testing and Fault Injection

Testing. The purpose of the Main Engine TCM is to test the hardware and software under a realistic set of circumstances where the main engine is used to alter the path of the spacecraft, Precision Thrust Vector Control (TVC) is performed by the flight software and it is critical that the dynamic model of the actuators alter the thrust vector as commanded by the flight software. As the following data in figures 3.4 and 3.5 shows, the EGA simulators performed well in a closed loop environment. The X and Y components of the thrust vectors of the simulation(py_me_[0 and

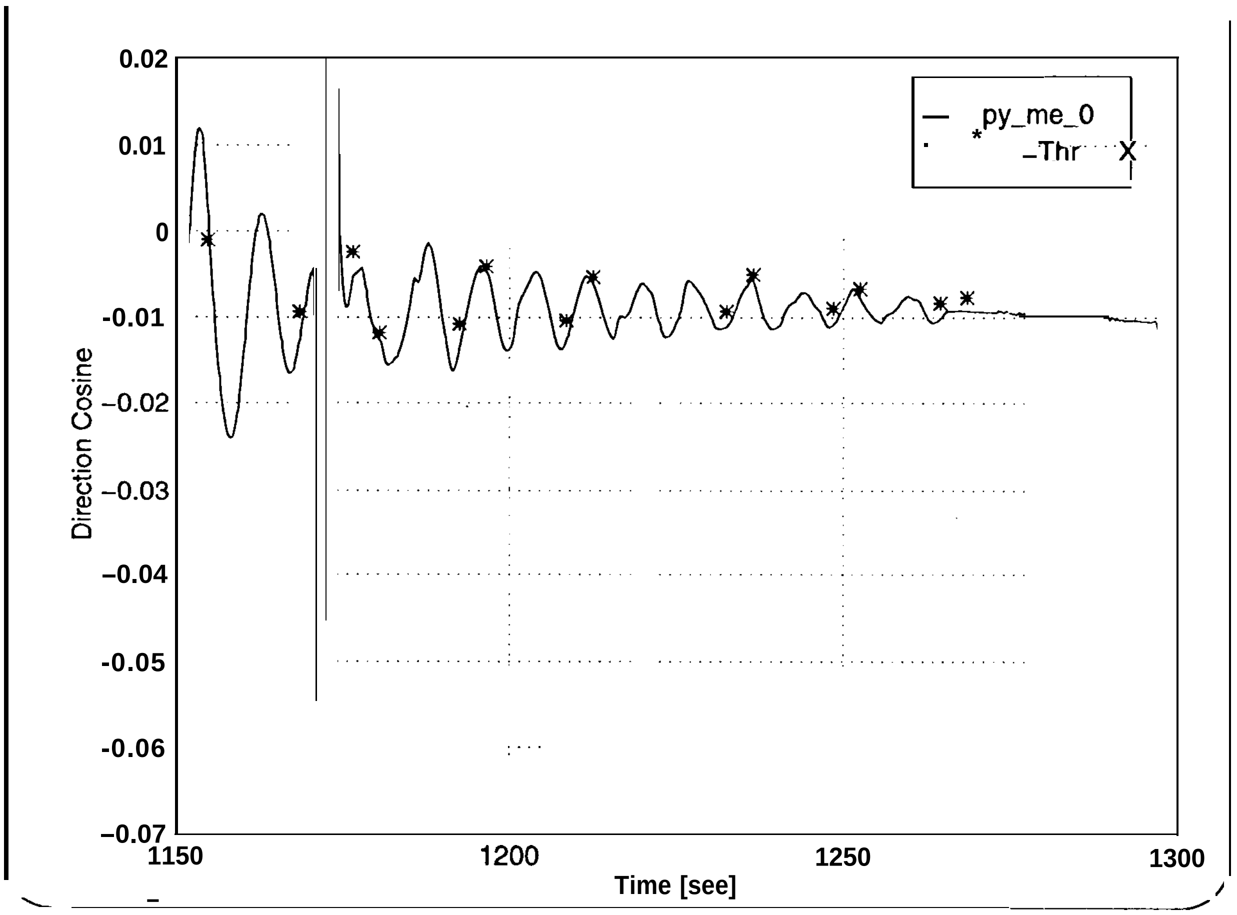


Figure 3-4 Simulated and Flight Software Thrust Vectors in the X direction

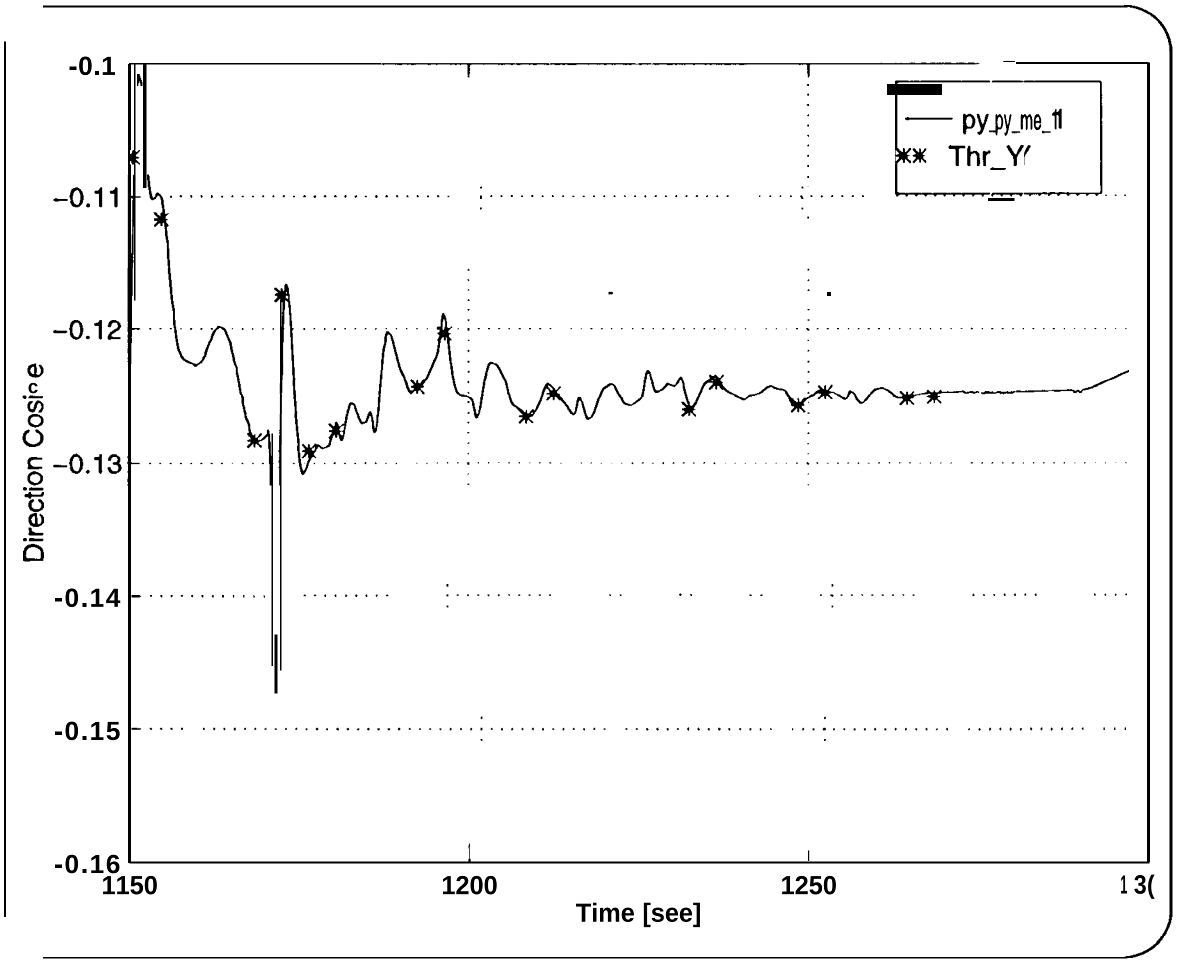


Figure 3-5 Simulated and Fight Software Thrust Vectors in the Y direction

1]) and what flight software believed to be the thrust vector (Thr_[X and Y]) matched very well. A Z component comparison is not available since the flight software does not record Z axis data since the direction cosine is so close to one.

Another closed loop aspect of the EGA simulation is the task of injecting faults into the simulation to test the fault protection responses. The EGA simulator was designed to interface with the support equipment exactly like the real hardware. Thus, it was not possible to simulate any faults that involved hardware failure. The software dynamics model could have been altered to simulate many faults, but since there is no way to alter the EGA hardware simulator there was no need. If the model was to simulate a stall, for example, it could easily be disabled from the rest of the simulation. This would not be possible for the hardware, however, and the result would be flight software receiving an external disturbance with no indication that a stalled EGA is the cause.

3.3 Evaluation

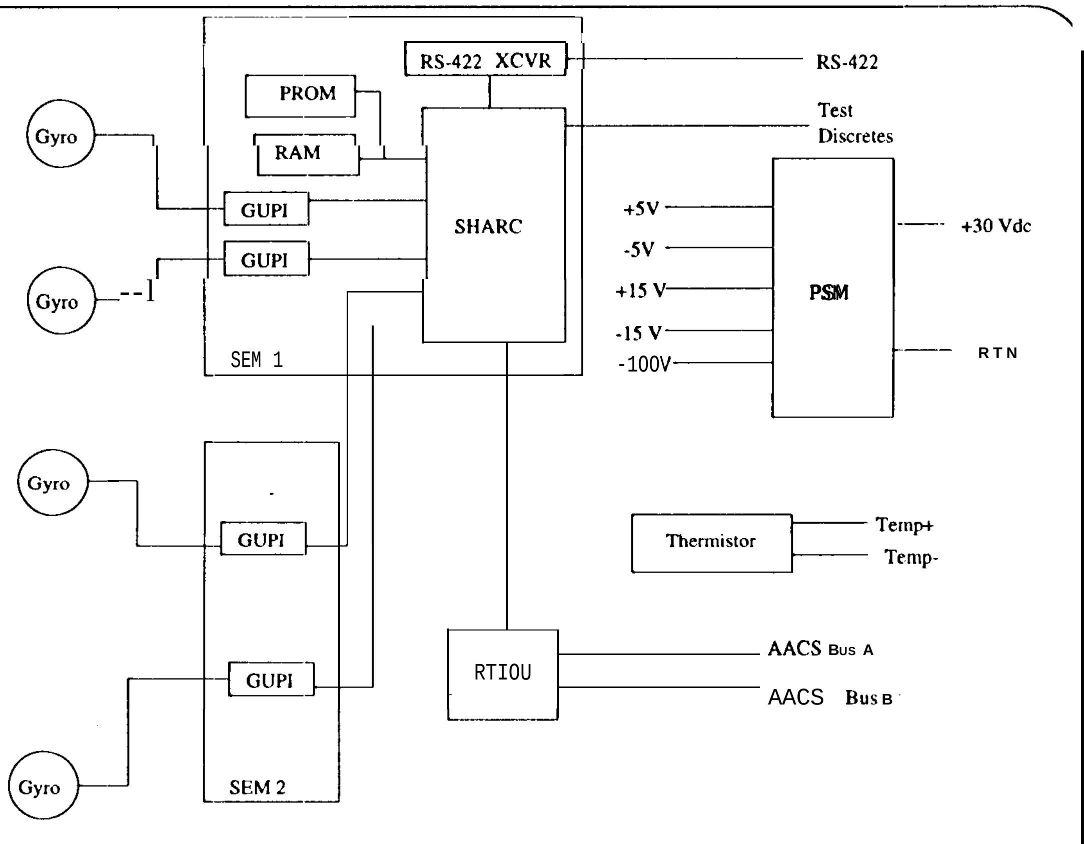
The EGA simulation met the requirement for test of the engine gimbal electronics. Specifically, the EGA simulator did not cause fault protection to activate unexpectedly during testing in either CATS or ITL. This facilitated testing of the flight software, as well as the Main Engine Trajectory Correction Maneuver. Due to the nature of the hardware simulator, testing of the fault protection capabilities was not performed. When the flight actuators were integrated with the AACS and the flight PMS, fault protection did activate unexpectedly. But since the EGA simulators were never meant to simulate a loaded EGA, the conclusion is that the EGA simulator did perform adequately during AACS testing.

Chapter 4

Inertial Reference Unit

4.1 Description

The Inertial Reference Unit (IRU) is a dual redundant assembly that is used to detect inertial angular velocity of the spacecraft. Each IRU contains four hemispherical resonating gyroscopes and processing electronics. The IRU contains its own processing circuitry that interfaces with the AACS databus RTIOU and the gyroscopes. A block diagram of the IRU is shown in figure 4. 1.[6]



SEM	Sensor Electronics Module	Circuit Card
SHARC	Space HRG and Accelerometer Readout Controller	Main processor IC (located on SEM)
GUPI	Gyro Unit Processor Interface	Gyro signal interface IC (located on SEM)
RTIOU	Remote Terminal Input Output Unit	External bus interface IC (located on SEM)
PSM	Power Supply Module	Power converter circuit card

Figure 4-1 Inertial Reference Unit Block Diagram

The gyroscopes each run at a frequency of approximately 2000 Hz and a rate estimate is obtained from the gyroscope every cycle. The SHARC processor has software that runs at 100 Hz. Each time the software loop is executed, the rate measurements since the last time the software was executed are integrated and the angle is added to the data that will be passed to” the flight computer. The flight computer can read the accumulated angle or the IRU status from the IRU. The AFC can also write data to the IRU. This data, for example, would be new software to the SHARC in the event of an IRU reset.

4.2 Cassini Laboratory Configuration

As in the case with the EGA simulation, there are two parts to the IRU simulation- an assembly simulator and a dynamics model. The assembly simulator is software that represents the IRU when the actual hardware is unavailable in the laboratory. The dynamics model is used to generated biases for the IRU by converting the spacecraft angular rates to rates in the gyro sensing axes which are used to bias either the real gyros or the simulator.

4.2.1 IRU Assembly Simulator

Description

The assembly simulator has the function of accepting biases from the dynamics interface and converting these biases into data for the flight software to interpret. This simulator consists of three parts. The first is a remote terminal input output unit. This unit is a remote terminal on the Cassini AACS databus and facilitates communication with the AACS flight computer. The second component is an interface card that communicates with the RTIOU. This card is a series of static RAM registers that the simulator can write to. This allows the software to act as the actual hardware by interfacing with the RTIOU similar to the actual hardware. The third component is the actual software for the IRU simulator. The software supports five functions of the IRU: power on initialize- -tion of the IRU output data, IRU Built in Test (BIT), IRU SHARC software download, IRU “soft reset”, and normal operation of the IRU. The data flow through these five functions are shown in figure **4.2.**

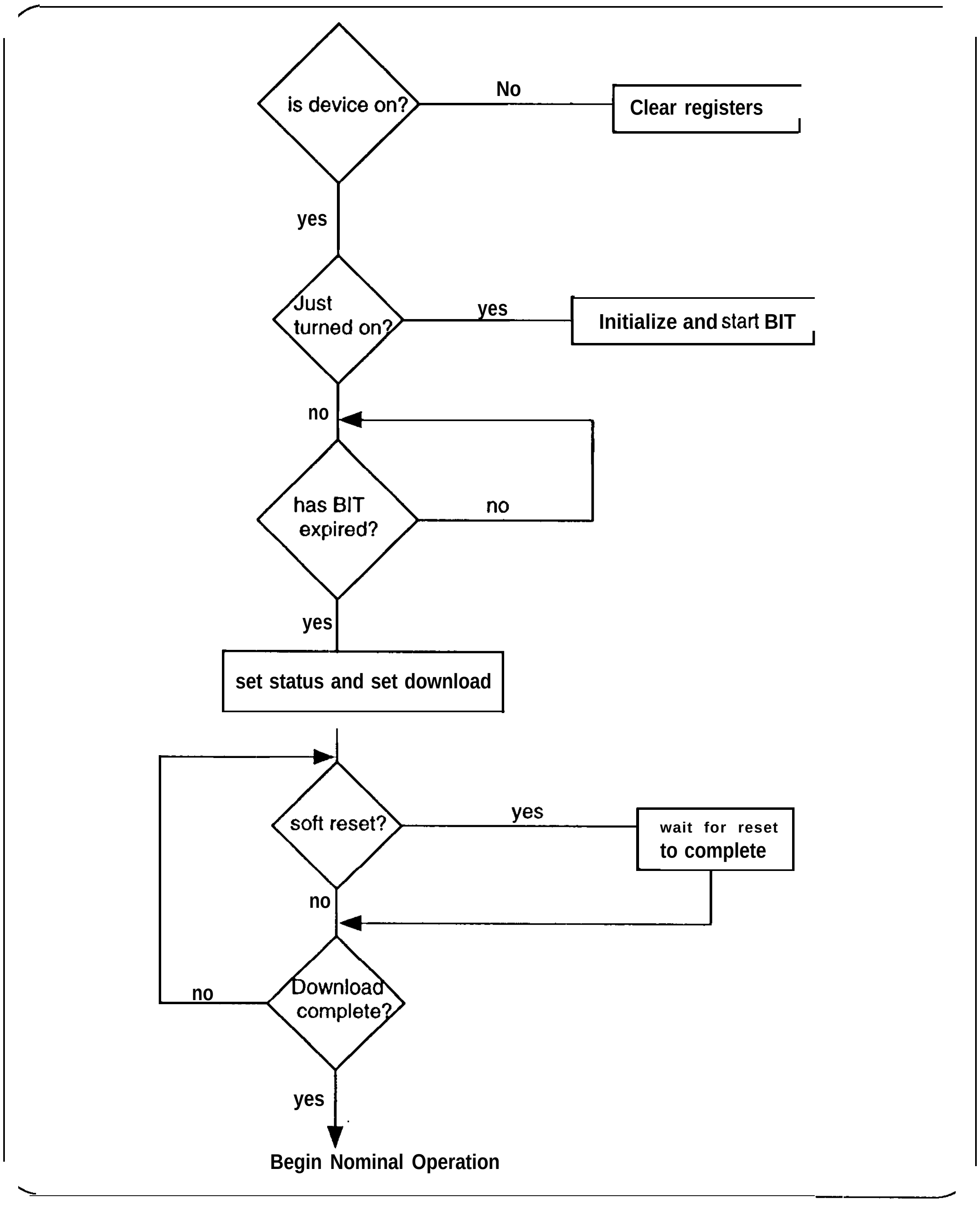


Figure 4-2 IRU Assembly Simulator Flow Diagram

Upon receipt of a power on command, the simulator sets all output information to zero and sets a timer to begin the BIT. The BIT simulation is a one second hold that simulates the time for the SHARC built in test. Once this hold is complete, the IRU simulator reports this information via the RTIOU to the flight software and sets a flag indicating readiness to begin the software download. The software download function is simulated by verifying receipt of the data sent by the flight software and then indicating a valid checksum and a valid load. The soft reset is another one second hold for the simulator and results in reporting a good status message back to the flight software after completion of the hold.

Finally, there is the actual operation of the IRU. The software checks for a soft reset and if none occurs, the software proceeds to calculate the outputs of the gyroscopes. Since the IRU simulator receives the rates the gyroscopes sense in the gyroscope coordinate system, the simulator simply has to convert these rates into a change in angle and properly format this information for the RTIOU. This is accomplished by multiplying the rate received by the IRU cycle time and adding this angle to the last angle computed to determine the accumulated angle. Once this is accomplished, the angle is converted to a form compatible with the RTIOU and the data is passed to the flight software.

Validation Testing and Results

This simulator was integrated into the ITL and CATS with the same integration procedure that was used for the actual flight hardware. Thus, we have a common test to compare the simulator and the hardware with. During the integration procedure, the gyros are biased with a set of support equipment commands. Figures 4.3and 4.4 show the result of these commands.

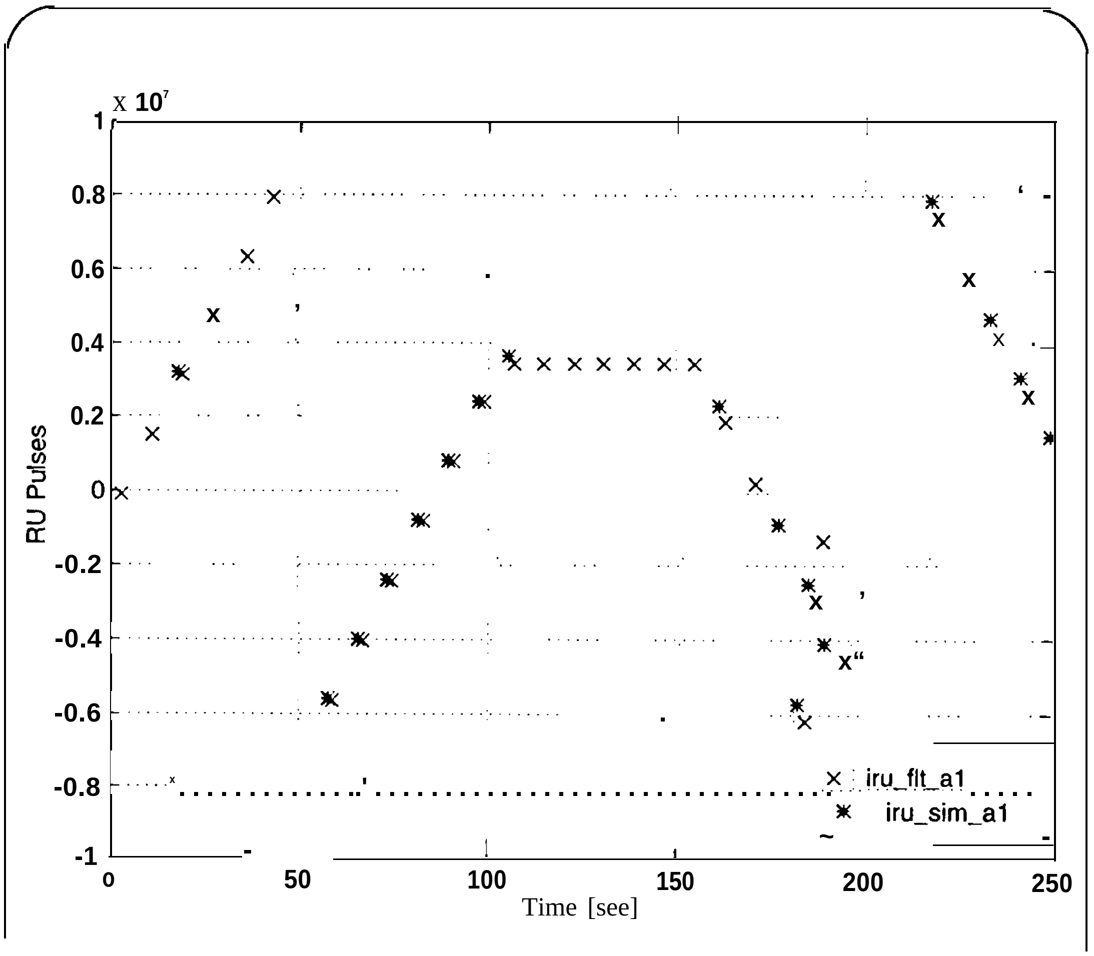


Figure 4-3 IRU Simulator and Hardware Comparison

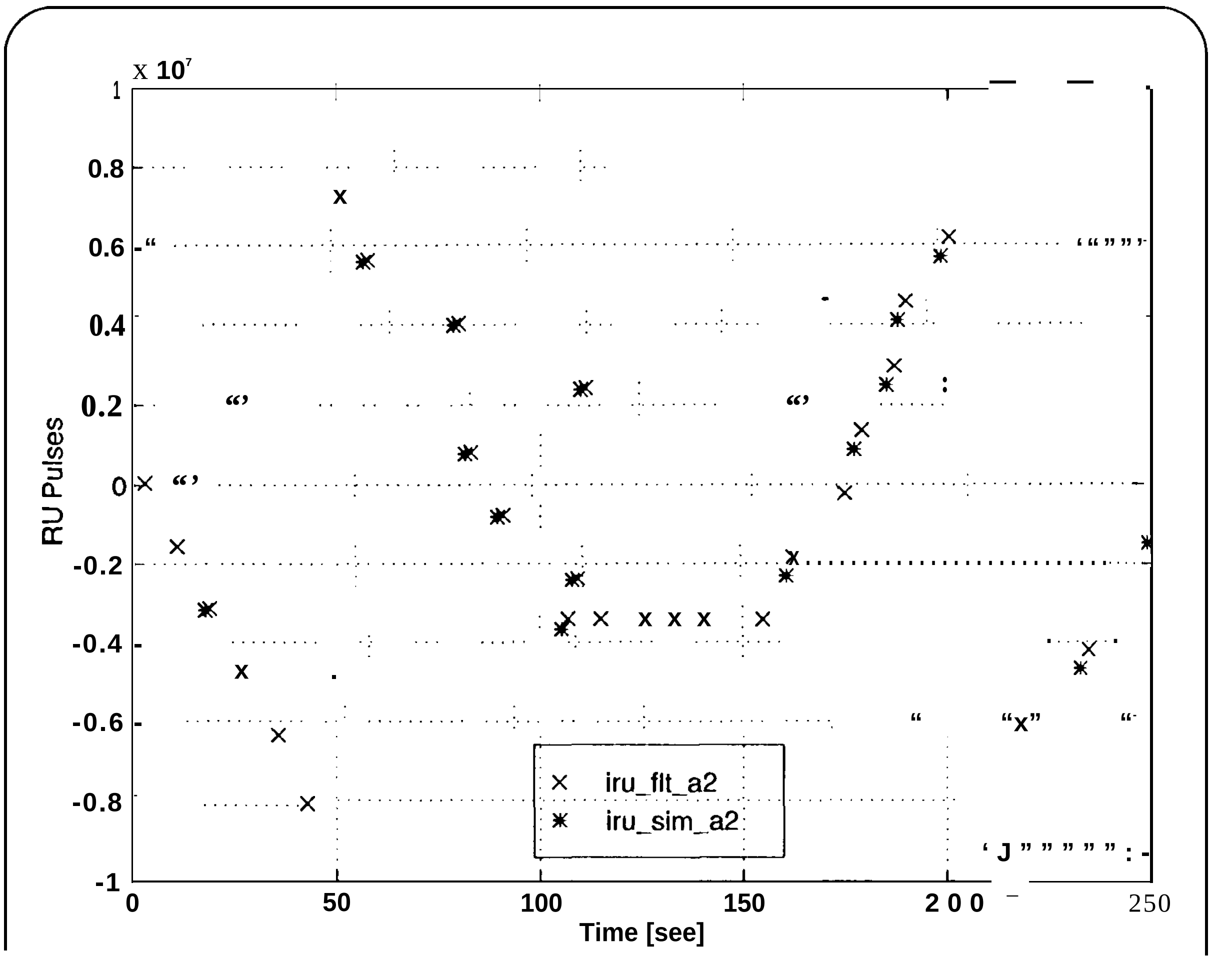


Figure 4-4 IRU Simulator and Hardware Comparison

As the data shows, the IRU simulators matched the performance of the flight equipment very well.

Differences and Problems

Integration revealed some problems with the simulator that have been corrected to increase the fidelity of the software. The first problem was with the processing cycle. The original version of the software integrated at the same speed as the dynamic simulation, 16 Hz. This is not consistent with the actual hardware, which reads data from all four gyros at a speed of 100 Hz. There is the danger that the simulator was running too slow and unrealistically large changes in angles would be reported by the simulator. It would be more consistent with the hardware if the software ran at a speed of 400 Hz and processed one gyroscope at a time. When all four gyros were processed, the simulator made this data available to the flight software through the RTIOU. This configuration change was implemented and performs very well.

A second problem was with the noise of the IRU. The original model did not have any simulation of gyroscope noise and once again a change would improve fidelity. The final solution was for the model to cycle through a large set of experimental data representing realistic numbers for gyroscope noise. These numbers would be added to the output of the gyroscope and would simulate noise coming from the IRU gyroscopes,

A final problem was with the timing of the simulator. In reality, the IRU has a double buffer that is used to prevent the flight software from reading partially updated data, One of these buffers always has a complete packet of information for the flight software. However, the assembly simulator card does not support double buffering. Thus, flight software was reading the information out of the IRU simulator, but the information was not completely updated. To correct this problem, the assembly simulator card was modified such that when flight software was reading the information from the card, a signal was sent to the IRU assembly simulator. The software was changed to verify this signal was not active before a read was attempted. If this signal was sensed, output was delayed until the read was concluded, This correction worked very well and the information to the flight software was valid.

The general conclusion is that once coding errors are corrected, the simulator does act sufficiently like the hardware to permit testing. However, there are several aspects of the hardware that are not simulated. Essentially, the actual computation of the SHARC is not simulated. The Built In Test is not actually performed. Instead, a timer simulates the delay the BIT would cause in the processing cycle. Similarly, the checksum during the download of the new SHARC software is not performed. Again, a timer is used to simulate the delay of the download and checksum. This was done because simulation of the actual SHARC software was not an objective in the IRU simulation. The basic philosophy was that the inputs and outputs of the IRU would be as identical as the software team could make them to the hardware. Thus the simulator can report that the data has been received or that a BIT has been performed, but the BIT or data download may not have actually happened in the simulation software. This black box concept has important consequences when fault injection is considered.

One of the advantages of using simulators is that the test analyst has the ability to simulate faults without damaging actual hardware. However, with the IRU simulator, any faults to the SHARC or other processing electronics cannot be simulated. The simulators were designed such that the "outside world" only "has a command path to the simulator if it has a command path to the actual device. Therefore, since one cannot command a BIT failure with the actual hardware, for example, it was decided that one would not be able to with the simulators either. Any fault injection would have to be possible with either real hardware or simulators and thus involve changing existing inputs into the devices. For example, it was possible to simulate a failed gyro by sending a large bias into one of the gyros. The SHARC or IRU simulator would see this large rate on one of the gyros and declare that gyro's data invalid. But failures internal to the IRU would have to be simulated on another testbed.

4.2.2 **Gyro Dynamics Model**

The gyro model within the dynamics simulation computes the biases that are sent to the IRU assembly simulator model. The model is straightforward. First, the locations of the gyroscopes and the transformations of the angular rates of the spacecraft from spacecraft to the gyro sensing axes were determined and loaded into the model. Then the model accepts the spacecraft angular rates from the DARTS simulation and converts these to rates that the gyroscopes sense. Then this information is passed to the IRU model as described above, The interaction between the model and the simulation is shown in figure 4.5.

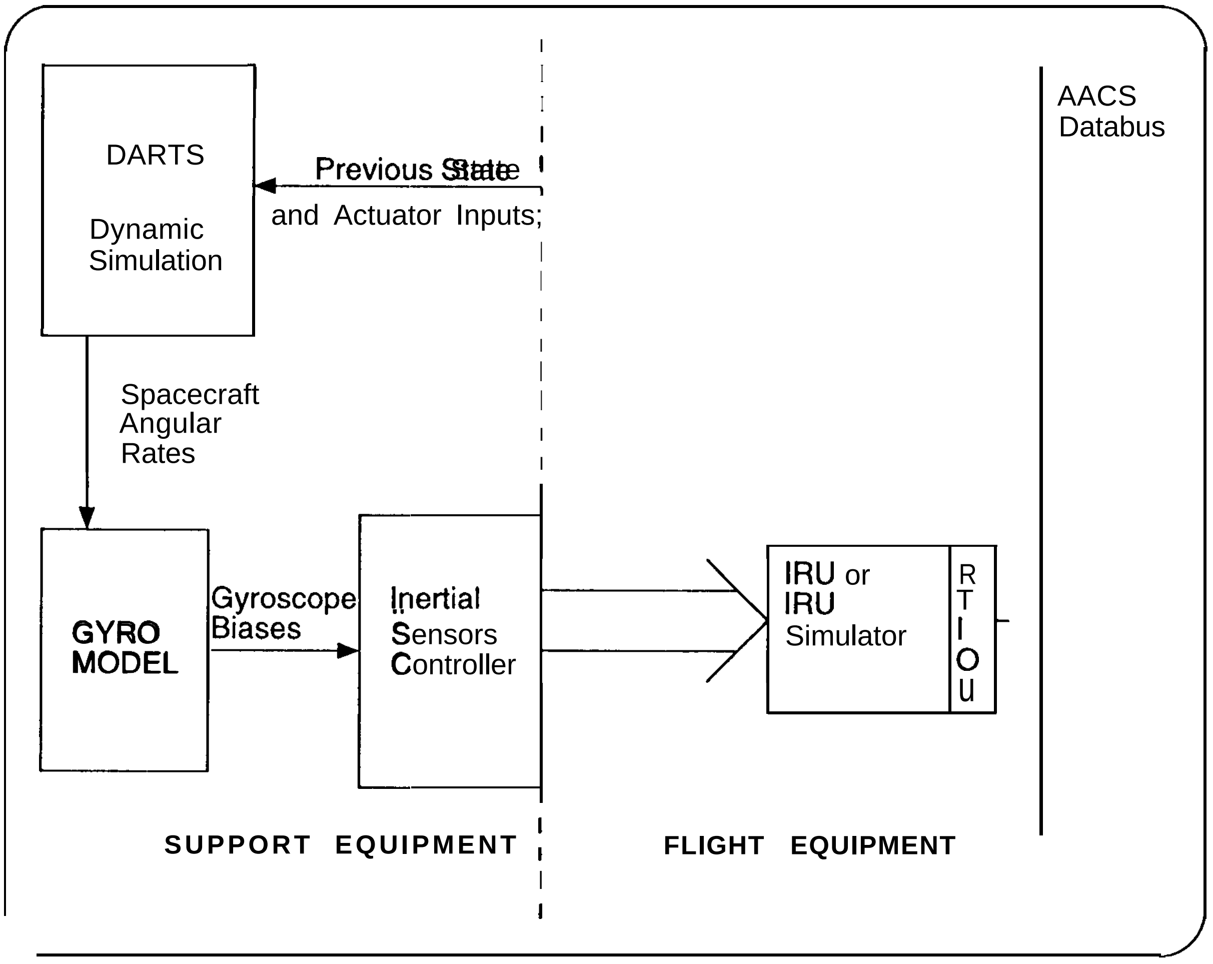


Figure 4-5 Inertial Reference Unit Dynamics Model

The gyroscope dynamics model was validated as a part of the closed loop simulation described below.

4.2.3 Closed Loop Simulation

The inertial reference unit simulator was used extensively in all laboratories during the testing of the Cassini AACS. The criterion for success of the simulator is how effectively the simulated angular rates of the dynamics simulation are reported to the flight software. This can be shown by examining the reported spacecraft rate and the simulated rate. This is shown in figures 4.6 through

4.8. the variable “**b_ang_rt_[x,y** or z]” is the angular rate being simulated and the variable “**[X,Y**

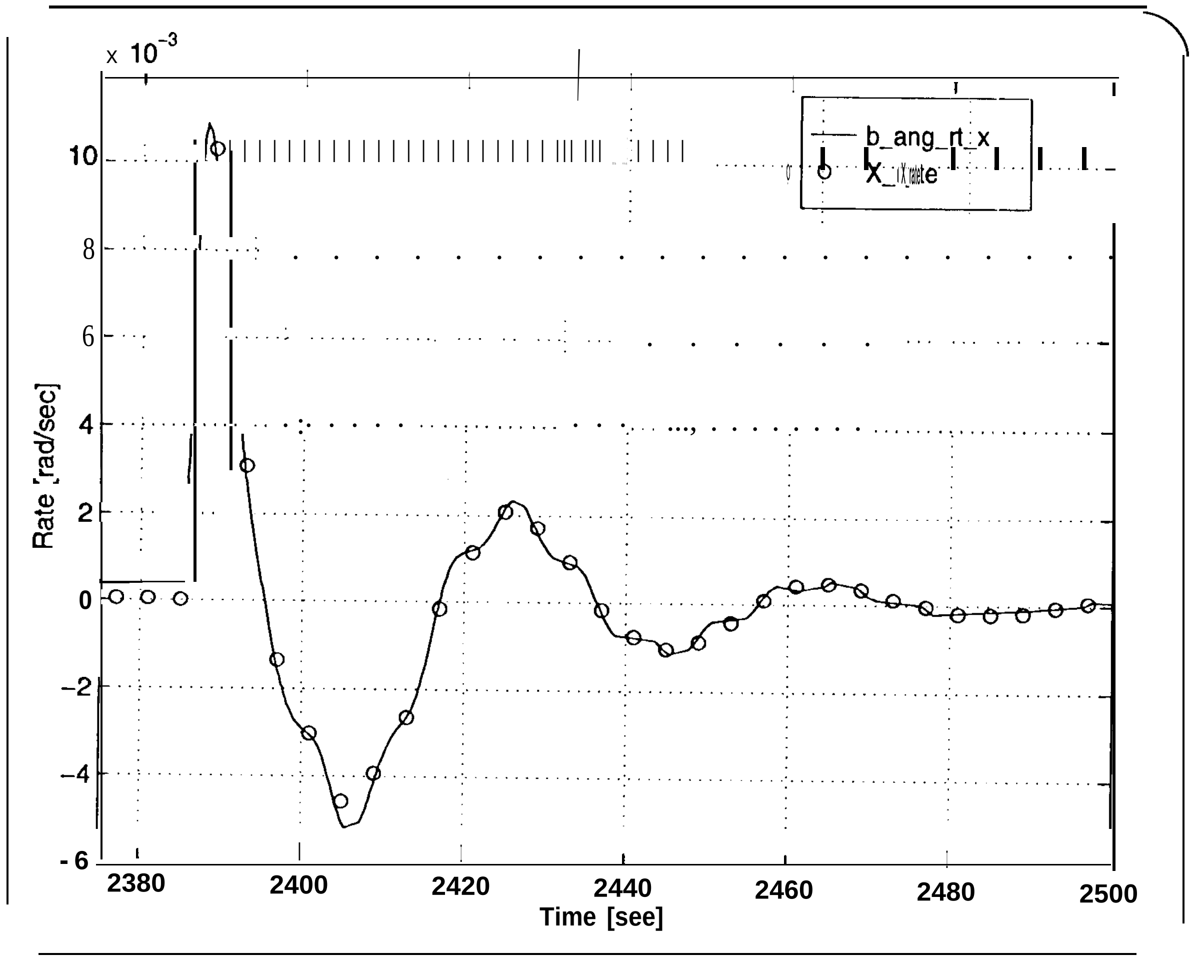


Figure 4-6 Simulated and Flight Software Angular Rates, X Axis

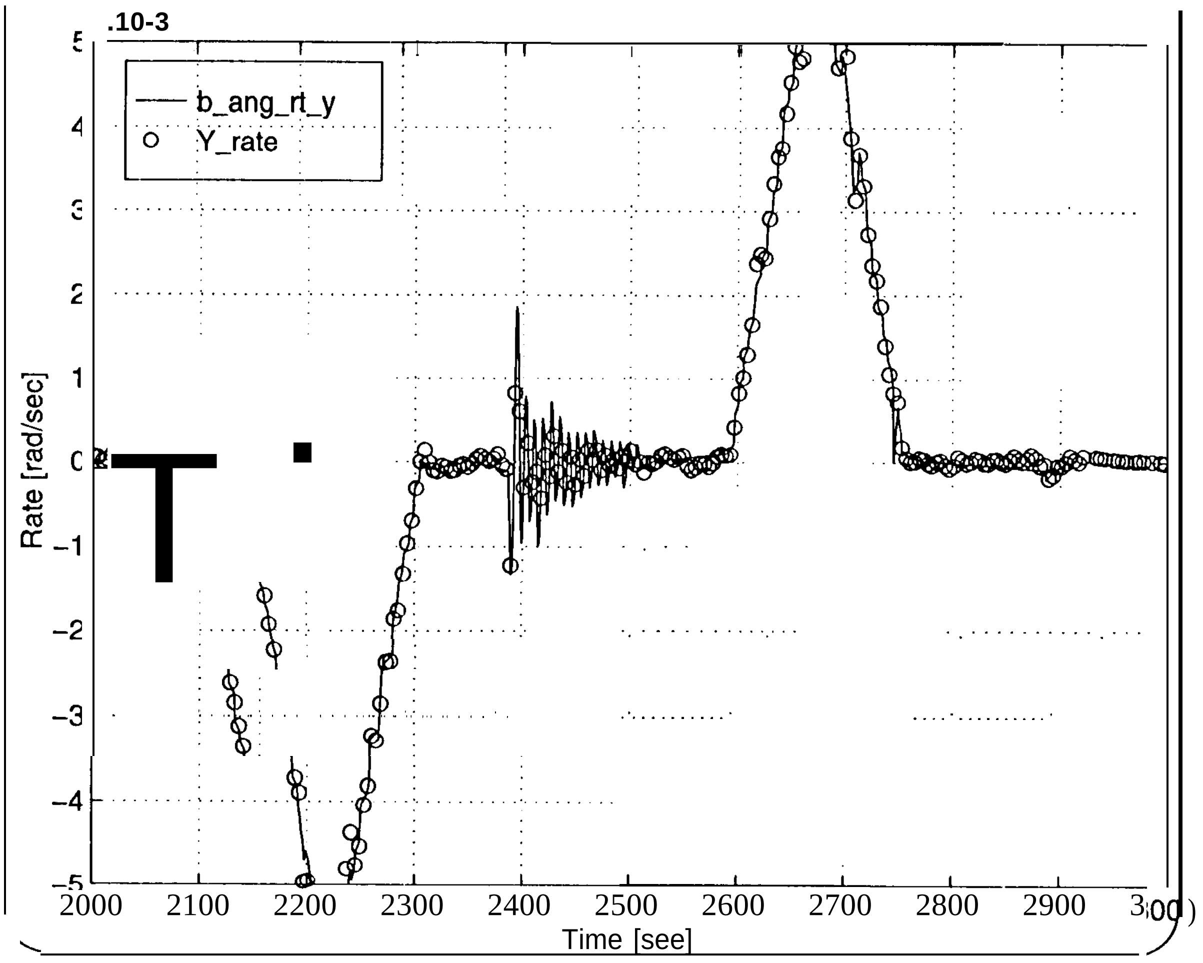


Figure 4-7 Simulated and Flight Software Angular Rates, Y Axis

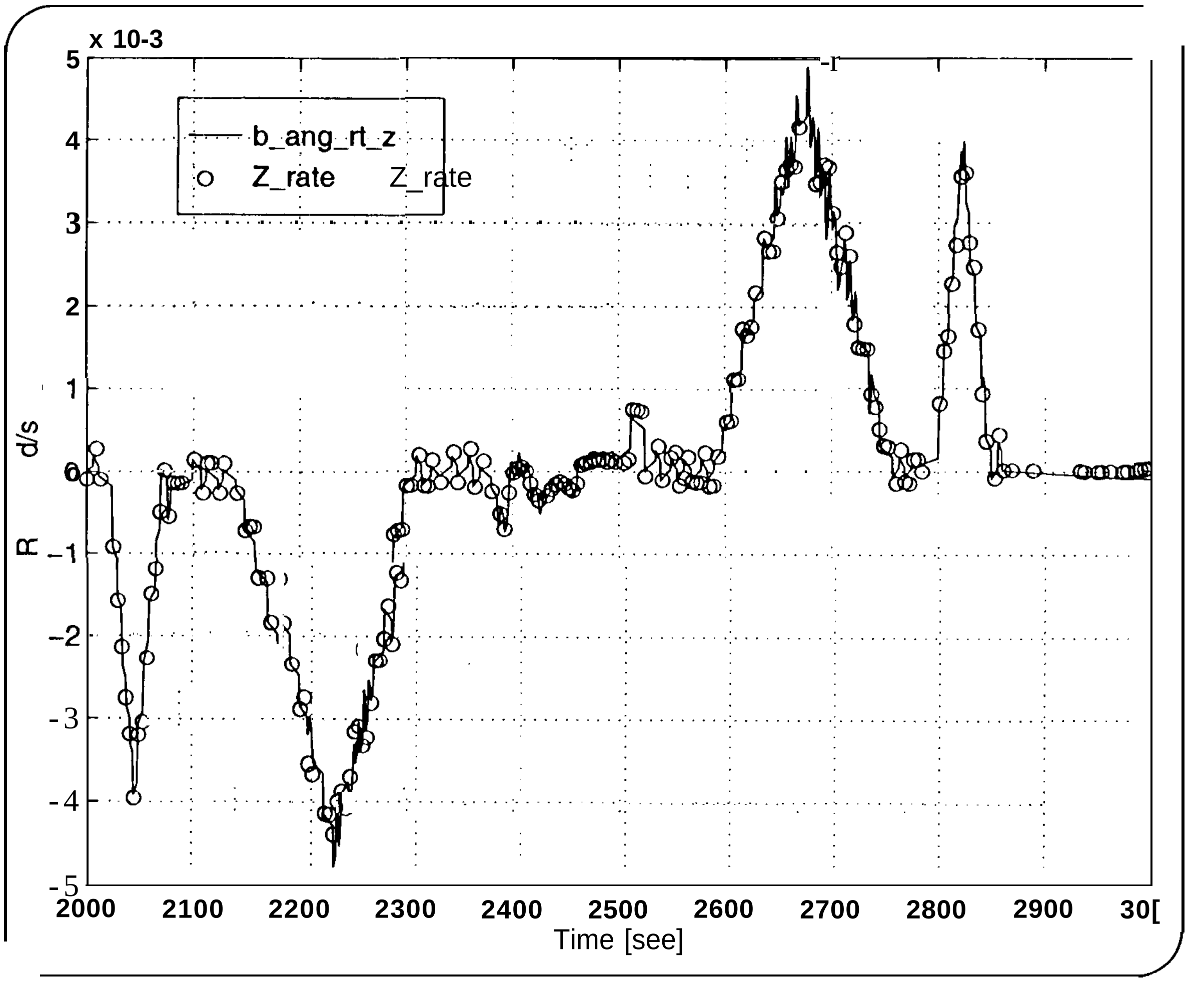


Figure 4-8 Simulated and Flight Software Angular Rates, Z Axis

or Z]_rate" is what flight software is reporting the angular rate to be after converting data from the IRU simulator. Both sets of rates track extremely well and we conclude that the IRU simulator performs well in the closed loop environment.

4.3 **Evaluation**

The Inertial Reference Unit simulator has performed very well in all three laboratories and has proven a valuable tool when IRU hardware was unavailable. The integration and test of the simulator revealed that the simulator had several shortcomings, but once these were corrected the simulator had a high degree of fidelity when compared to the actual hardware and closed loop performance during flight sequences have been excellent, The IRU assembly simulator represents the actual hardware well and, even though fault injection testing was limited, still performed adequately in ITL and CATS.

Chapter 5

Reaction Wheel Assembly

5.1 Description

The Cassini spacecraft hastwosources ofattitude control. The first source is the Propulsion Module Subsystem consisting of the main engines and 16 reaction control thrusters. The second form of control is the reaction wheel assembly. There are four reaction wheels on the spacecraft, three of which are the primary reaction wheels and the fourth which is a backup reaction wheel. The purpose of the reaction wheels is to store angular momentum of the spacecraft as well as to provide attitude control.

The reaction wheels receive commands from the flight software over the AACS databus. There are eight commands that are accepted by the reaction wheels. These are summarized in table 5.1 below. [6]

Table 5.1: Commands and responses for the Reaction Wheel Assembly

Command	Response
Read Delta Angle	Return the accumulated angle count, (including over (under) flow indicator) and reset counter to zero.
Read Torque	Return the current command torque setting for the RWA
Read RWA Current	Return the current value of total RWA electrical current.
Read Motor Current	Return the current value of RWA motor electrical current
Read Status	Return the current operational status data of the RWA
Set Torque	Hold the reaction torque output at the value specified in the command
Reset	Set delta angle pulse counter to zero before resumption of counting, and set torque command to zero. This response shall be automatically executed at the time power to the RWA assembly is commanded on.

The reaction wheels utilize a brushless DC motor to spin the wheels and a hall effect tachometer to sense the motion of the wheel. Twenty four magnets are attached to the reaction wheel and hall devices sense the motion of these magnets pasta sensor. This sensor then is the tachometer which increments by one each 1/24th of a revolution of the reaction wheel.

5.2 Cassini Laboratory Configuration

5.2.1 RWA Assembly Simulator

Description

As in the case of the IRU, the RWA assembly simulator consists or a remote terminal input output unit for the RWA, software for the simulation, and an interface card that permits communication between the two. The reaction wheel assembly simulation software is a C program that simulates the dynamics of the reaction wheel for the purpose of calculating outputs to communicate with the AFC via the RTIOU. The inputs to the program area torque command from the AFC and power on/off commands. The program's outputs are tachometer counts, wheel power, and a torque command wrap around. The program data flow consists of reading the power and torque commands, propagating a three dimensional state vector consisting of the wheel position, rate and a time dependent frictional term, and then computing the tachometer and power outputs for the AFC.

When the loop starts, the model reads a RAM register to determine if the AFC has commanded the wheel to power on. If the wheel is off, the power state is set to zero and the state continues to propagate. In that case, the model would simulate frictional spin down of the reaction wheel. If the wheel power state is set to one, the model performs two more reads of the RAM registers to read the torque enable command and the 2's complement torque command.

The second step of the loop is to propagate the state of the system. This is performed by computing the derivative of the state and then performing numerical integration to determine the actual state. To determine the derivative of the state, the program first computes the total torque that will be applied to the wheel. This torque computation is shown in figure 5.1 and consists of four elements.

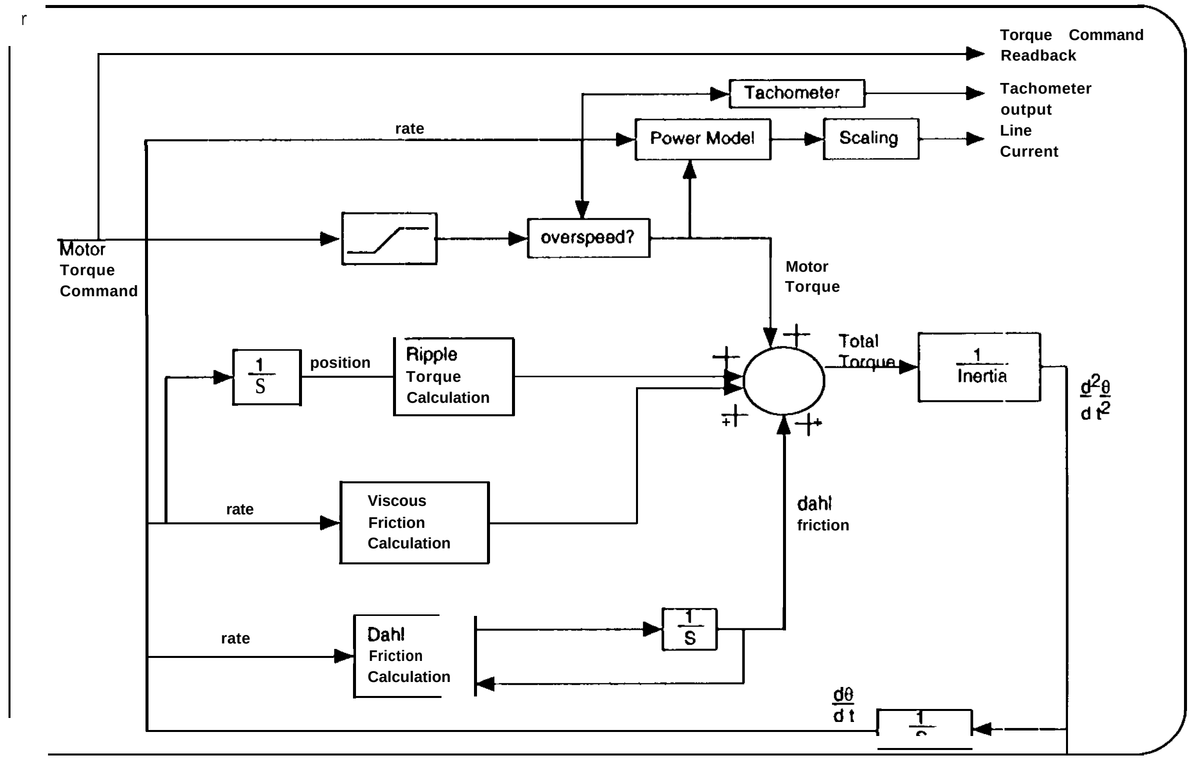


Figure 5-1 Reaction Wheel Assembly Simulator Torque Computation

First, there is the commanded torque. This is converted from two's complement to an equivalent value in Newton Meter-seconds and compared against a maximum motor torque. If the commanded exceeds the maximum, the motor torque is set to the maximum, There is also an overspeed flag that is set if the wheel is spinning too fast, If it is spinning too fast, the motor torque is set to zero. The second element of the torque command is a term to account for commutation ripple. This torque is due to the brushless DC motor and is sinusoidal in nature due to the switching of the DC current windings.

The third component of the torque is the viscous friction which opposes wheel velocity and is determined by multiplying the wheel rate by a constant. Finally, there is bearing friction to consider. This torque is called Dahl friction based on the bearing model developed by P.R. Dahl. [1] This model computes the time derivative of the friction and is integrated to determine the torque to apply. These four components are summed to determine the torque that will be applied to the wheel.

Once the torque is known, the state derivative is calculated. This is accomplished by setting the derivative of the position to the rate and the derivative of the rate to the torque divided by the wheel inertia. The derivative of the Dahl friction term is calculated via Dahl's model, These values are then used in a 4th order Runga Kutta numerical integration algorithm to determine the state of the system. The time step used in this routine in 0.0625 seconds.

Finally, the output is computed. This is performed by first calculating the power consumed by the wheel based on the vendor's power model and converting the power into a current. Next, the tachometer data is computed. This simulation runs every 62.5 ms, but the tachometer data is read by the flight software every 125 ms. Therefore, the tachometer output should reflect what the real tachometer would say after 125 ms. This is accomplished by multiplying the current rate of the wheel by 125 ms. This gives an estimate of the position of the wheel after 125 ms. Then, this is multiplied by a scale factor representing the quantization of the tachometer. This is then the output of the tachometer. Since the output is a integer, the fractional value computed by this calculation is saved and added to the next read. In this manner, no tachometer counts are lost. Finally, the current, tachometer counts, and the torque command are sent to the RAM registers for transmission through the RTIOU back to the AFC.

Validation Testing and Results

The reaction wheel assembly simulator was unit tested by performing the reaction wheel simulation integration procedure in the lTL, CATS, and ALTO. This procedure contains power off and power on tests to verify electrical interfaces between the AFC and the RWA simulators. The power off section is performed with break out boxes in the loop to protect the hardware in the event of a incorrect connection. The power on tests verify that the flight software can command the wheels and that the reaction wheel simulation responds as expected. These tests were performed in the ITL and in CATS and no significant problems were found that would inhibit testing using the simulators.

A second set of tests was also performed. This set duplicated tests that were performed on the flight equipment during stand alone testing. These tests were performed in CATS to allow for comparison of simulator response and that of the actual hardware. The results of this test is shown in figures 5.2 and 5.3. Figure 5.4 shows the error between the two rate plots.

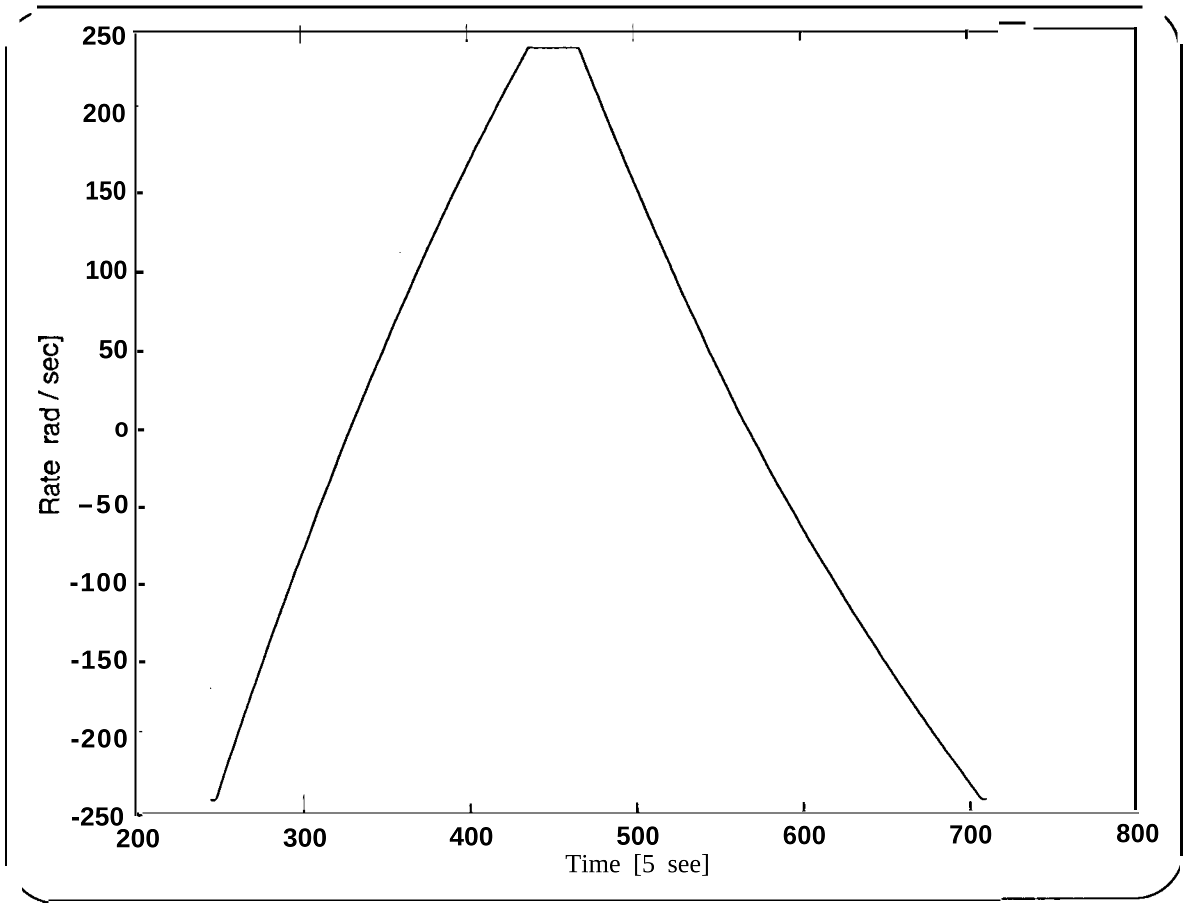


Figure 5-2 Reaction Wheel rates during comparison test

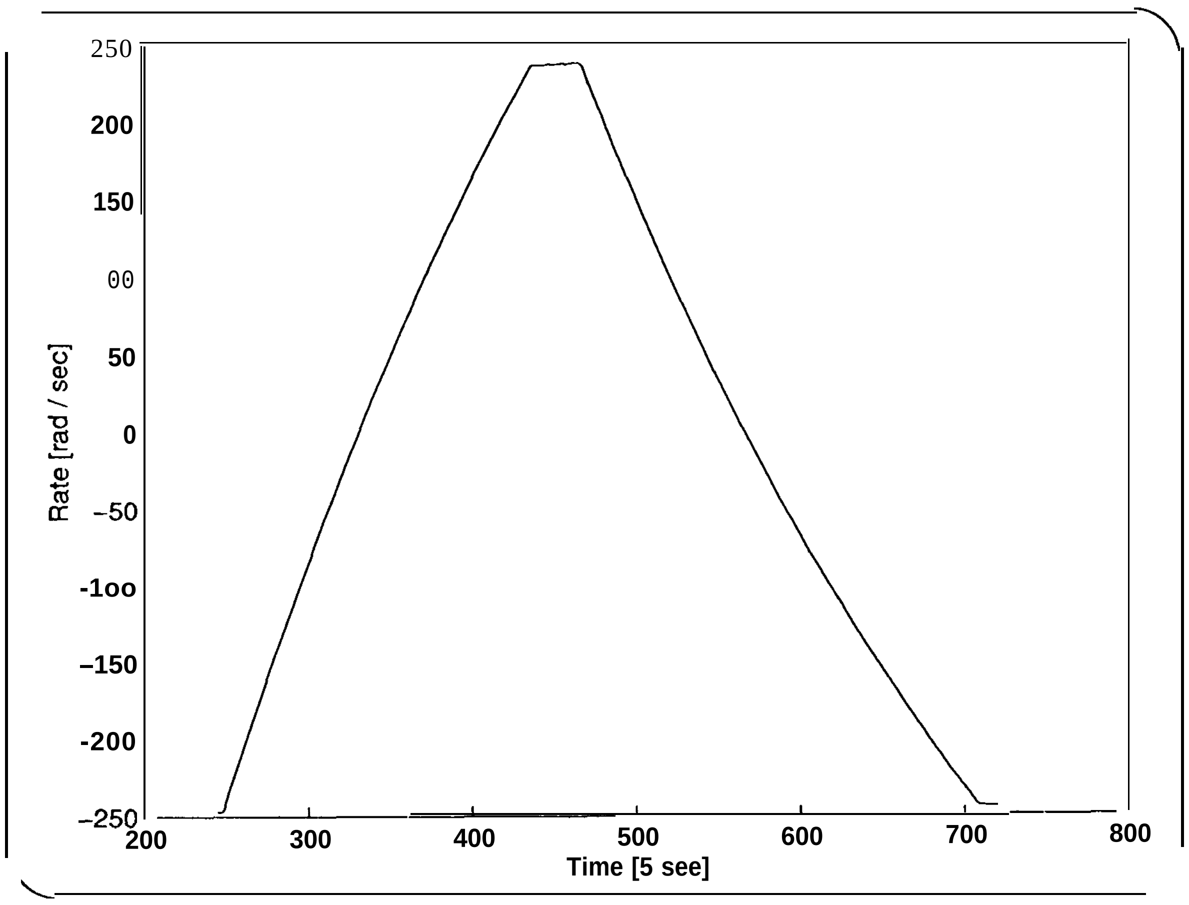


Figure 5-3 Reaction Wheel simulator rates during comparison test

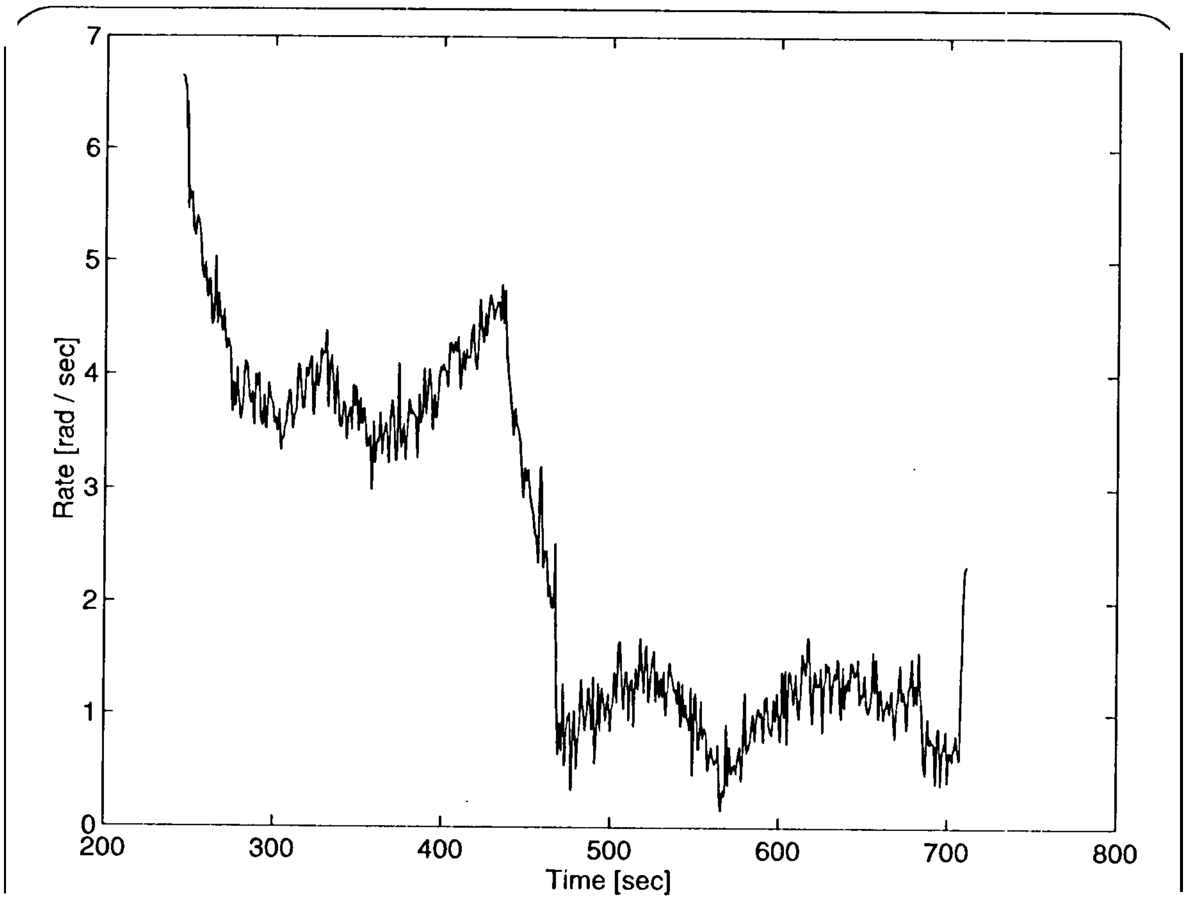


Figure 5-4 Error between reaction wheel simulator and hardware

As is shown in the plots, particularly in the error plot, the simulator was able to track the hardware quite well. The initial error is essentially a result of not being able to exactly reproduce the earlier test. The hardware was tested in a stand alone environment, while the simulator was tested with flight software active. As a result, the simulator test was subject to flight software constraints on allowable wheel torques and rates. However, during wheel rate changes on the order of 500 radians per second, the error was less than 10 radians per second. In addition, this error did not change appreciably when the wheels were accelerating.

There are a number of important differences between the simulation and the real reaction wheels. The first difference is the multiplexer that exists on the real wheel. This allows the flight software to receive different measurements of reaction wheel current and voltage. However, flight software is not designed to use any information other than the line current for the reaction wheels. Thus, the simulator only computes line current.

A second difference points to an important issue that was seen earlier with the IRU simulator. Neither of these simulators have any knowledge as to when the flight software is going to read the RTIOU output. This means that information must be kept current at all times at the RTIOU output. A consequence of this is seen in the way the output of the RWA was computed. The RWA routine runs at 16 Hz, but the flight software reads the RTIOU of the RWA at 8 Hz. Thus, it would be desirable to run the output routine at 8 Hz as well. The problem is that if this is attempted, the simulator and the flight software diverge from each other and the result is incorrect output. “Old” information is kept at the RTIOU registers for too long and incorrect information is relayed to flight software. This is why the output routine runs at 16 Hz and integrates the tachometer output as if it was running at 8 Hz. This way, the simulator and the flight software will not diverge enough to cause problems. In addition, the reaction wheel simulator will use the signal from the assembly simulator card indicating a flight software read to prevent data corruption.

A third difference lies in the fact that the static ram that allows communication between the software and the RTIOU uses the same registers for reading and writing, That is, the memory location for reading information from a register is the same that is used to write information to that register. This is a problem with register 5 of the RWA. This register is simultaneously the lower byte of the tachometer data from the RWA simulator and the load mux command to the RWA. This means that there is a chance, if the timing is unfortunate enough, for the RTIOU to read what it believes is the lower byte of tachometer output, but is actually the old load mux control command from that last cycle. This is a rare occurrence due to the faster processing time of the RWA simulator, but if the processing takes excessively long or if flight software is quicker than usual, bad data could reach the AFC. With the real RWA, read and write registers are separate, but this is not the case with the simulator and thus this danger exists.

One of the most difficult features of the reaction wheels to simulate was the response of the reaction wheel to a reset of its Remote Terminal on the AACS databus. A reset of the remote terminal input/output unit (RTIOU) can be occur one of three ways: the RTIOU can be commanded to reset, it resets as a result of a power on, or the RTIOU can reset due to a timeout. A timeout occurs when the RTIOU does not receive commanded for 250 ms. An RTIOU reset results in the reaction wheel setting its torque command to zero until further instructions are received from flight software. This prevents the reaction wheel from acting on an incorrect or obsolete torque command and spinning out of control. Furthermore, the reaction wheel hardware manager inside flight software expects to read a zero torque value back from the reaction wheel and will not command the wheels until this zero is received.

The problem with simulating this reset response is threefold. First, the static RAM card that allows the reaction wheel simulation software and the RTIOU to communicate does not include a channel to relay the RTIOU reset information to the reaction wheel assembly simulator. Therefore the software had no direct knowledge of the RTIOU reset. Secondly, the static RAM uses the same memory locations as read and write registers. This means that if an RTIOU reset was detected falsely and the software wrote a zero to the torque memory location, good torque commands would be overwritten. The final difficulty is that the flight software and the reaction wheel simulator have no knowledge of each others timing. A reset could occur anytime during the simulators processing cycle and the software would have to somehow correctly write a torque command of zero when a reset occurs.

The first attempt to solve this problem is depicted in figure 5.5. The software nominally runs twice

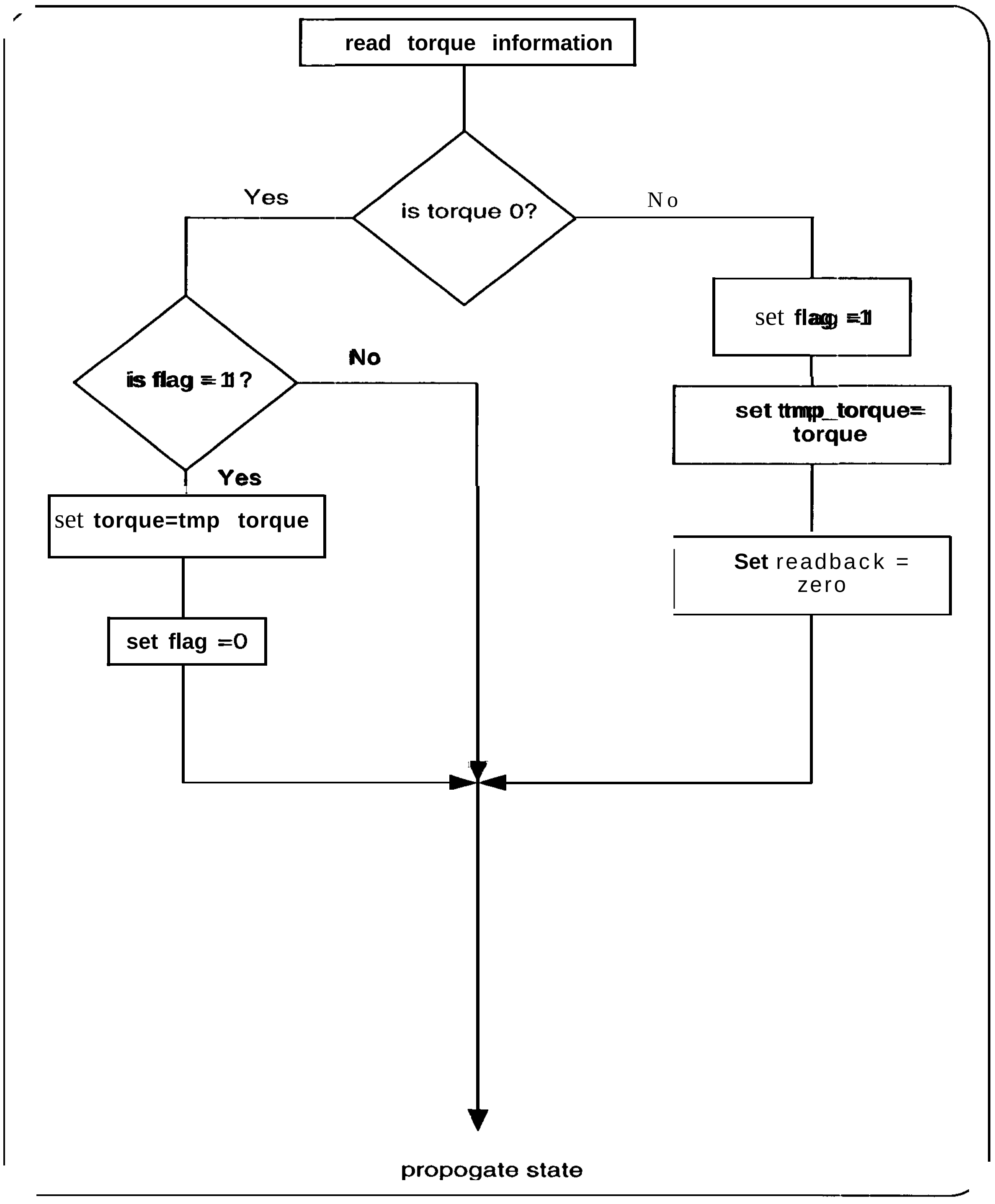


Figure 5-5 First attempt in solving remote terminal reset anomaly

as fast as the flight software, so the simulator should expect a new torque command every other cycle. The first action is to read a new torque command and overwrite the RAM memory location with a zero. Then the simulation continues using the read in torque command. It also copies the command to a temporary buffer. The next cycle, the software expects a zero (which it wrote) since no new update has occurred. If this is true, the software acts on the previous torque command that was buffered and propagates the state again. Finally, the register is read a third time. If the zero is still present, the software assumes that a reset has occurred since it did not receive a new update and thus acts on the zero until a new update is received, when the cycle begins again.

This method did not work well. The software overrode the good torque commands with zeros incorrectly and the simulator failed. This was caused by the fact that flight software timing is not exact and synchronizing the simulator and flight software was too difficult in a real time environment. This can be seen be examine the following databus transactions

```
96-264/1 8:30:52.153 bm_d_rwx2 = 000eae0407074001004006 65 c096

96-264/1 8:30:52.235 bm_s_rwx2 = 00 100706ae09eOOOOOffOOOd91 65 c072

96-264/1 8:30:52.278 bm_d_rwx2 = 000eae0407074001004006 65 c096

96-264/1  8:30:52.360 bm_s_rwx2 = 00100706ae09e00000ff000d91 65 c072

96-264/1  8:30:52.403 bm_d_rwx2 = 000eae0407074001004006 66 c094

96-264/1 8:30:52.548 bm_s_rwx2 = 00 100706ae09eOOOOOffOOOe89 00 c093

96-264/1 8:30:52.590 bm_d_rwx2 = 000eae0407074001004006 00 c061

96-264/1 8:30:52.673 bm_s_rwx2 = 00100706ae09eOOOOOffOOOd89 00 c08f

96-264/1  8:30:52.715  bm_d_rwx2 = 000eae0407074001004006 65 c096

96-264/1 8:30:52.860 bm_s_rwx2 = 00100706ae09eOOOOOffOOOe89 00 c093
```

```
96-264/1 8:30:52.903 bm_d_rwx2 = 000eae0407074001004006 00 c061

96-264/1 8:30:52.985 bm_s_rwx2 = 00100706ae09eOOOOOff000d89 00 c08f

96-264/1 8:30:53.028 bm_d_rwx2 = 000eae0407074001004006 65 c096

96-264/1 8:30:53.110 bm_s_rwx2 = 00 100706 ae09eOOOOOff000e9 1 65 c076

96-264/1 8:30:53.153 bm_d_rwx2 = 000eae0407074001004006 66 c094
```

This data is from testing on GMT day 96-264 in CATS. Data denoted at bm_s_rwx2 are source bus packets for RWA2 (from RWA2) while those denoted as bm_d_rwx2 are destination bus packets for RWA2 (from the AFC). Appendix A has a complete RWA bus data decoder. the critical information is decoded and explained below.

The first source packet contains a OX66 (denoted by the spacing before and after) which is the torque command read back from the reaction wheel. (OX66 = 102 dn [dummy units]. This corresponds to a torque of 0.14 Nm) The next destination packet writes a 65 to register 6 (the torque register). This is acted upon and the RWA reports back a 65. This happens again with no incident. But observe what happens at 18:30:52.403. The flight software commands a torque of 66, but the RWA is delayed in responding until 18:30:52.548. Since the flight software commands a reading from the reaction wheels about 4 ms before the bm_s_rwx2 data appears, this corresponds to a delay between flight software commanding a torque and reading the reaction wheel of over 140 ms. This delay results in the simulator reporting a zero as a read back, since the software believed an RTIOU reset had occurred. Flight software responds by requesting a zero and verifying it reads back a zero. This happens at 18:30:52.673 and thus flight software commands a torque of 0x65 to resume processing. But again a delay occurs and the torque returns to zero. The simulator/flight soft ware combination break this pattern at 18:30:53.028 and nominal operation continues.

Bases on this information, it was deemed essential to support as many as three simulation iterations between flight software commands as well as to minimize the writing of zeros to the torque command. A new method was devised to met these additional requirements.

This new method is shown in figure **5.6.** This method uses the torque enable flag, which is set to

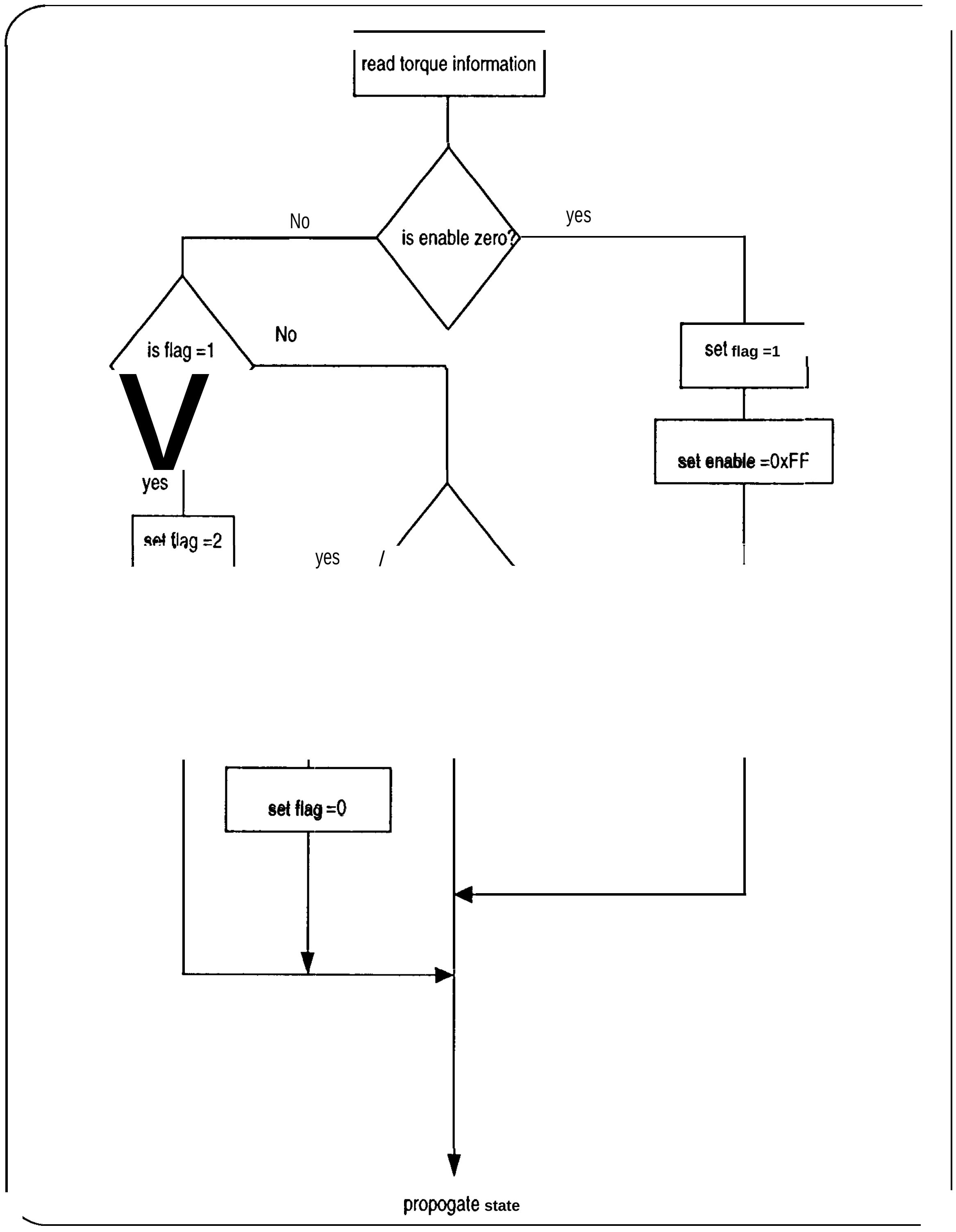


Figure 5-6 Second attempt to solve Remote Terminal Reset Anomaly

zero by the flight software each time a torque command is sent. The torque enable command is monitored at the beginning of each iteration. If the torque enable command indicates new data has been received, then the simulation accepts this data and resets the torque enable command. The simulation then acts on this new data until 3 cycles without an update are complete. Then the simulation software sets the torque command to zero and acts on it as well. Thus the torque is set to zero only if no updates are received for three cycles.

This method was much more successful and the results can be see in figure 5.7. The reaction wheel

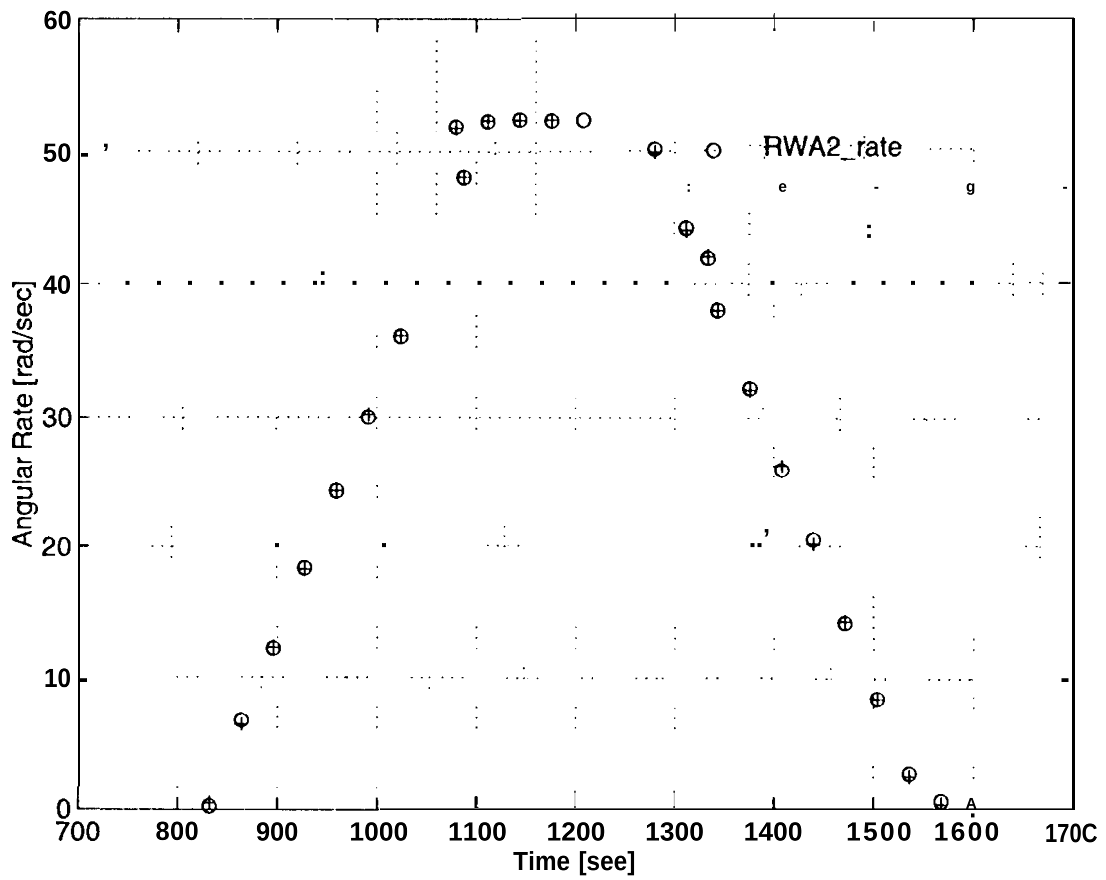


Figure 5-7 Successful restest of remote terminal reset anomaly

rates follow the idea rates computed by flight software very closely and the torque command remained constant even in the presence of an RTIOU reset.

This modeling revealed an important consideration in real time simulation. It is critical that the simulated wheels have the same access to information that the hardware does. The reaction wheel simulator does not have a reset line, nor does it have separate input and output registers. This lack of information made the RTIOU reset response a much more difficult item to model. This is an area for improvement for Cassini's Reaction Wheel simulator. The presence of a torque enable command was fortunate and allowed the software to model the reset response, but the software response is difference from the real hardware because the software and the hardware have different information available to them.

A final concern is the impact on fault injection. It has been decided that no fault injection would be conducted in the ITL or CATS with the RWA simulators. This is due to the same decision that impacted the fault injection of the IRU. Since no real command path exists to bias the tachometer of a real wheel, no attempt will be made to do the same with the simulator. In addition, the timing issues discussed above makes it impossible to guarantee that a particular flight software command will generate a particular response, so injecting faults such as an incorrect torque readback during a flight software cycle is not possible to simulate.

5.2.2 **RWA Dynamics Model**

Description

The reaction wheel dynamics model consists of a C program that is designed to simulate the reaction wheels in order to generate torques to apply to the spacecraft dynamics model. The model receives three pieces of information from the assembly simulator described in 5.2.1 above: the commanded torque which is read back from the assembly simulator, the tachometer reading from the assembly simulator and the time of the torque command read back. This information is used to drive the dynamics model as shown in figure 5.8. The first step is a call to DARTS to compute the angular rate of the modeled wheels. Once the rates are known, the model computes a rate error. First, the time since the last calculation is computed from the timing information. Second, a rate sample is calculated using the tachometer output and the elapsed time since the last calculation. This rate sample is put through a low pass filter that combines the rate sample with a previous rate estimate to obtain a new rate estimate. The purpose of this step is to limit the amount the rate estimate can jump in any one iteration. This rate estimate is then used with the rates of the DARTS wheels to obtain the rate error.

Once the rate error is known, the model computes the torque to be applied to the simulated wheels inside DARTS. First, a correction torque is computed. This correction torque is a function of an estimate of the drag torque and the rate error. The rate error is also used to compute the next value of the drag torque estimate. Finally, the torque applied is computed by subtracting the correction torque from the commanded torque.

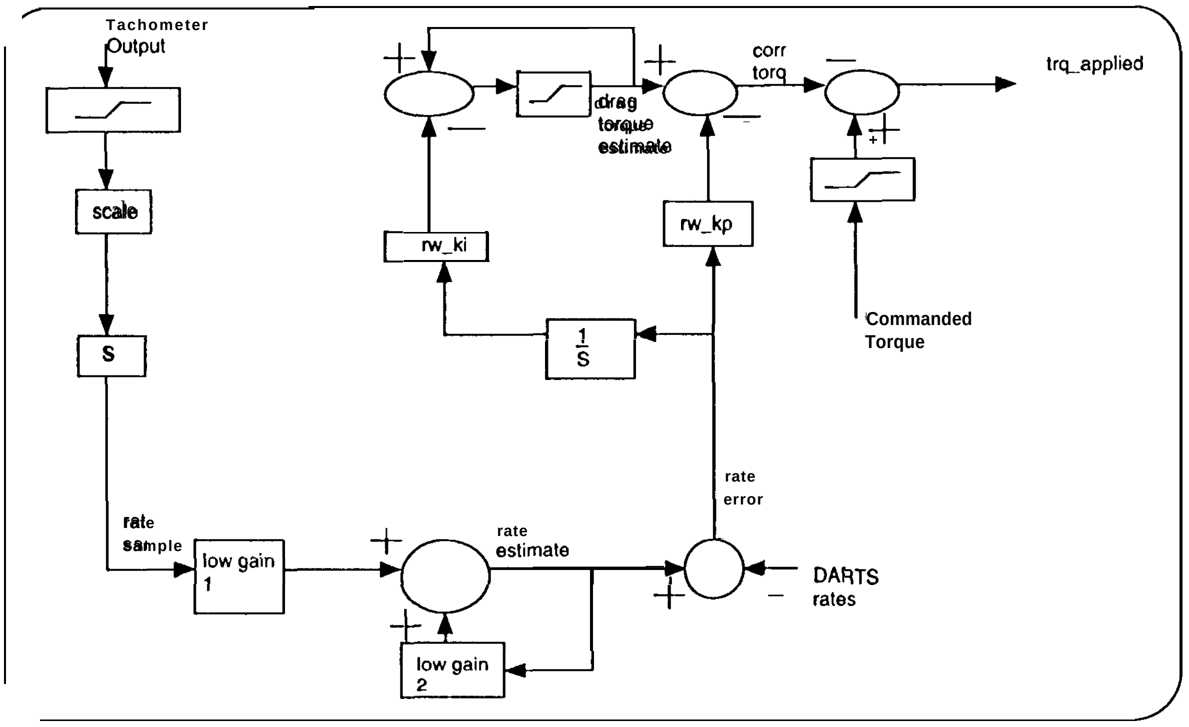


Figure 5-8 Reaction Wheel Dynamics Model Controller Block Diagram

Validation Testing and Results

To validate the model, conservation of angular momentum was invoked to develop predicts for the simulated spacecraft rates as follows:

$$H = I\omega \tag{5.1}$$

$$I_{XX,S/C}\,\omega_{X,S/C} = I_{re}\,\omega_{re}\,l_X \tag{5.2}$$

$$\omega_{X,S/C} = \frac{I_{rw}\,\omega_{rw}\,l_X}{I_{XX,S/C}} \tag{5.3}$$

Where

$I_{XX,S/C}$ = Principle Moment of Inertia, X axis, Spacecraft

$\omega_{X,S/C}$ = Angular Velocity about X axis, Spacecraft

I_{RW} = Moment of Inertia, Reaction Wheel

ω_{RW} = Angular Velocity, Reaction Wheel

1_x= Direction Cosine transform from reaction wheel to Spacecraft

Therefore, the effect of a reaction wheel's velocity on the spacecraft could be predicted, knowing the moments of inertia of the spacecraft and reaction wheel, as well as the transform from reaction wheel to spacecraft. The results of the testing was that after several coding errors were discovered and fixed, the model did match analytical predicts to within 1 e-4 radians/second. Model Validation uncovered one particular feature of the real time simulation that almost prevented the dynamical wheels from functioning before it was fixed. The problem was with the corruption of data coming to the simulator from the blackboard. The simulator assumes that the data it receives is valid and relies on this information to track the reaction wheels on the spacecraft side of the simulation (real or simulator). This means, for example, if the timetags used to determine the new speed of the spacecraft wheels are corrupted and the change in time is falsely computed to be extremely small, the model will compute an unusually large rate estimate and attempt to torque the dynamics wheel to match this incorrect rate estimate. This resulted in several problems that had to be corrected. For example, figures 5.9 and 5.10 shows data from GMT day 96-233 and 96-235 where

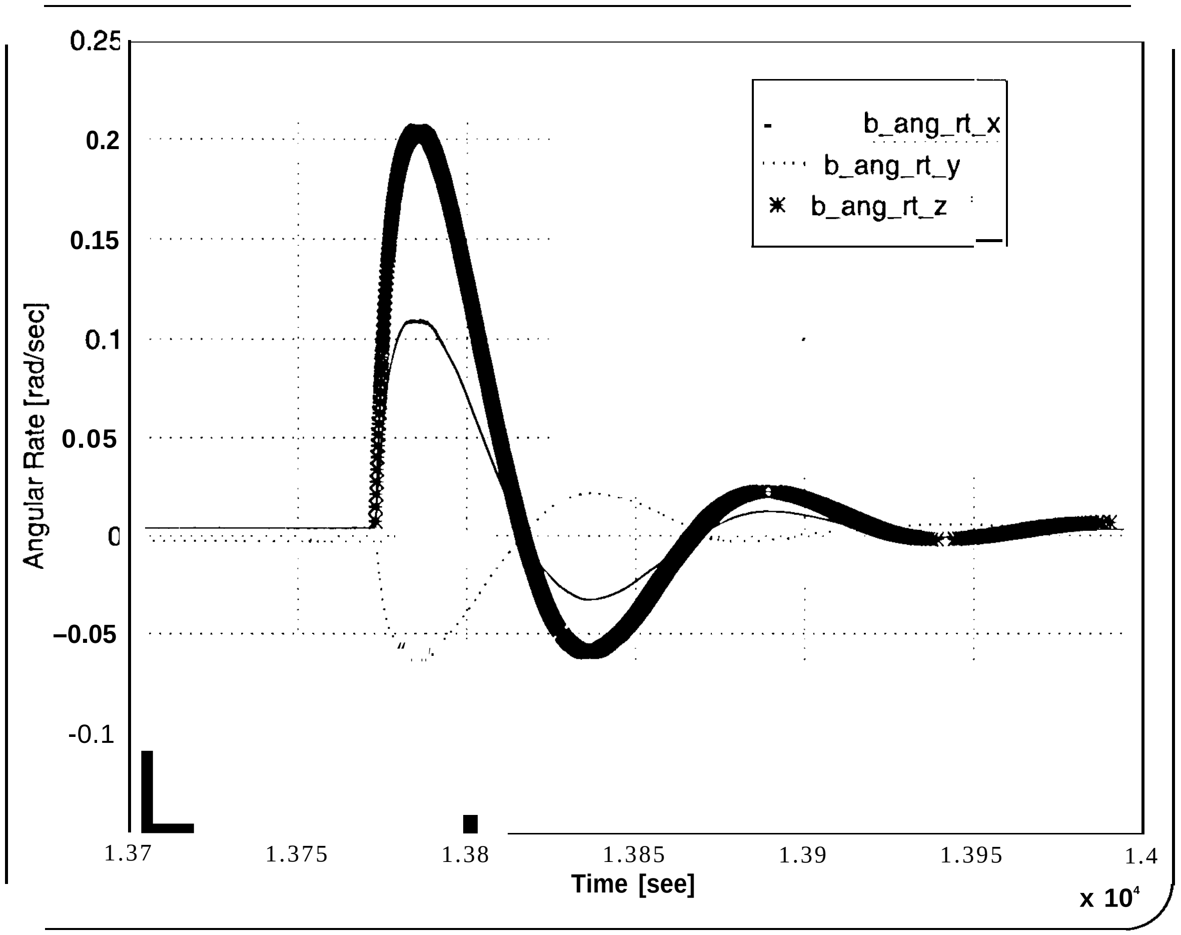


Figure 5-9 Reaction Wheel dynamics simulator anomaly- 96-233

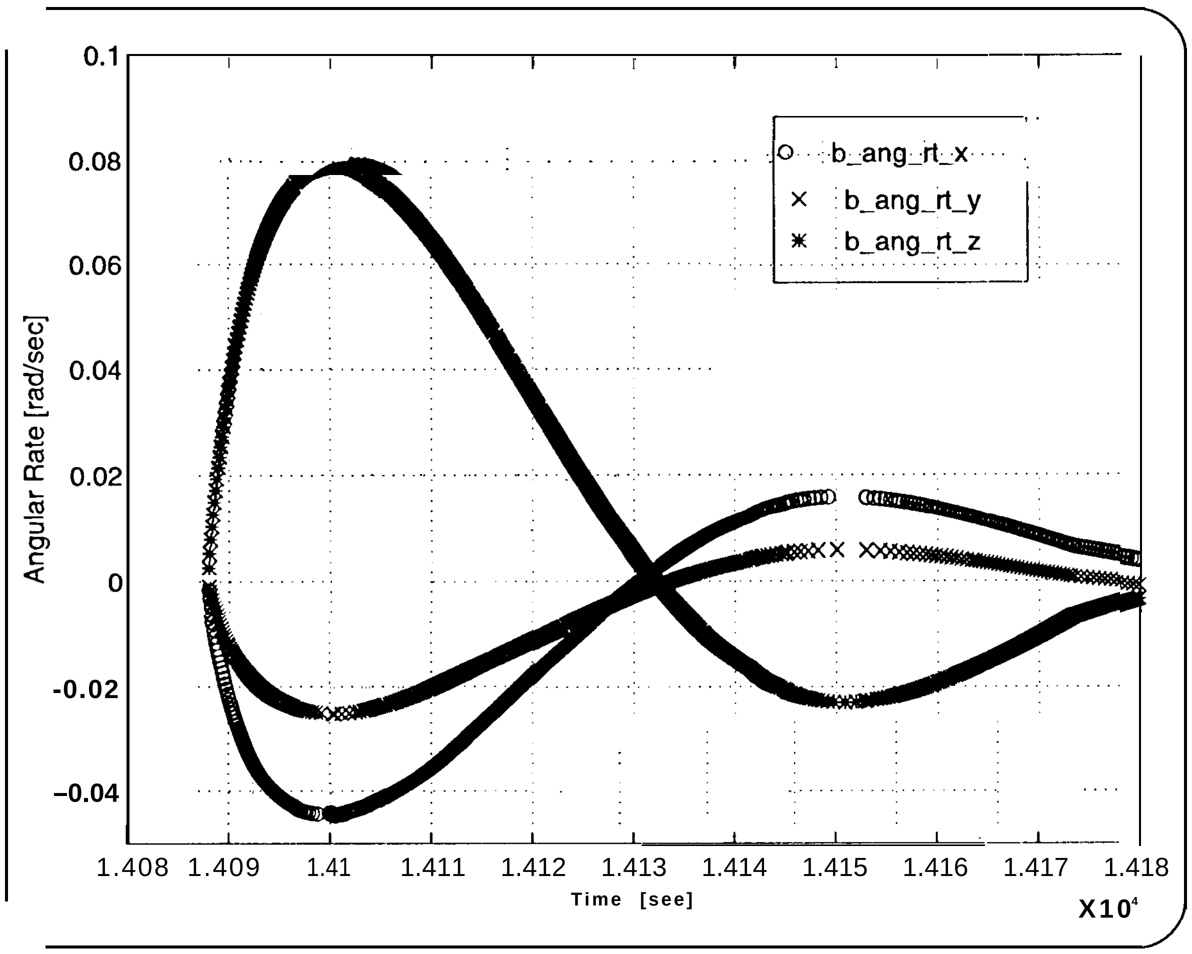


Figure 5-10 Reaction Wheel dynamics simulator anomaly- 96-235

the reaction wheels spun out of control and later recovered. Examination of the model software revealed that this second order behavior would be possible if the software received a small change in time estimate (used in the computation of the rate sample and the drag torque estimate). This would result in a torque sent to the dynamics wheels that would be unrealistically large based on the unrealistic speed and drag torque computed, and the rates would jump as shown. When the next update was reasonable, however, the rates would decrease and the wheels would gradually be brought under control. The response is second order due to the proportional plus integral controller used in the dynamics model. To "correct this, a limiter was added to the drag torque estimate (shown in figure 5.8). Filters were also added to the incoming tachometer and torque data, as well as software to limit the change in time estimate to within 25 ms of the expected value. Initially, the necessity of the limiter on the change in time estimate was not discovered and when the time is was not limited (and only the drag torque, torque and tachometer filters were activated), the error became a first order response, shown in figure **5.11.** This is because the drag torque filter limited

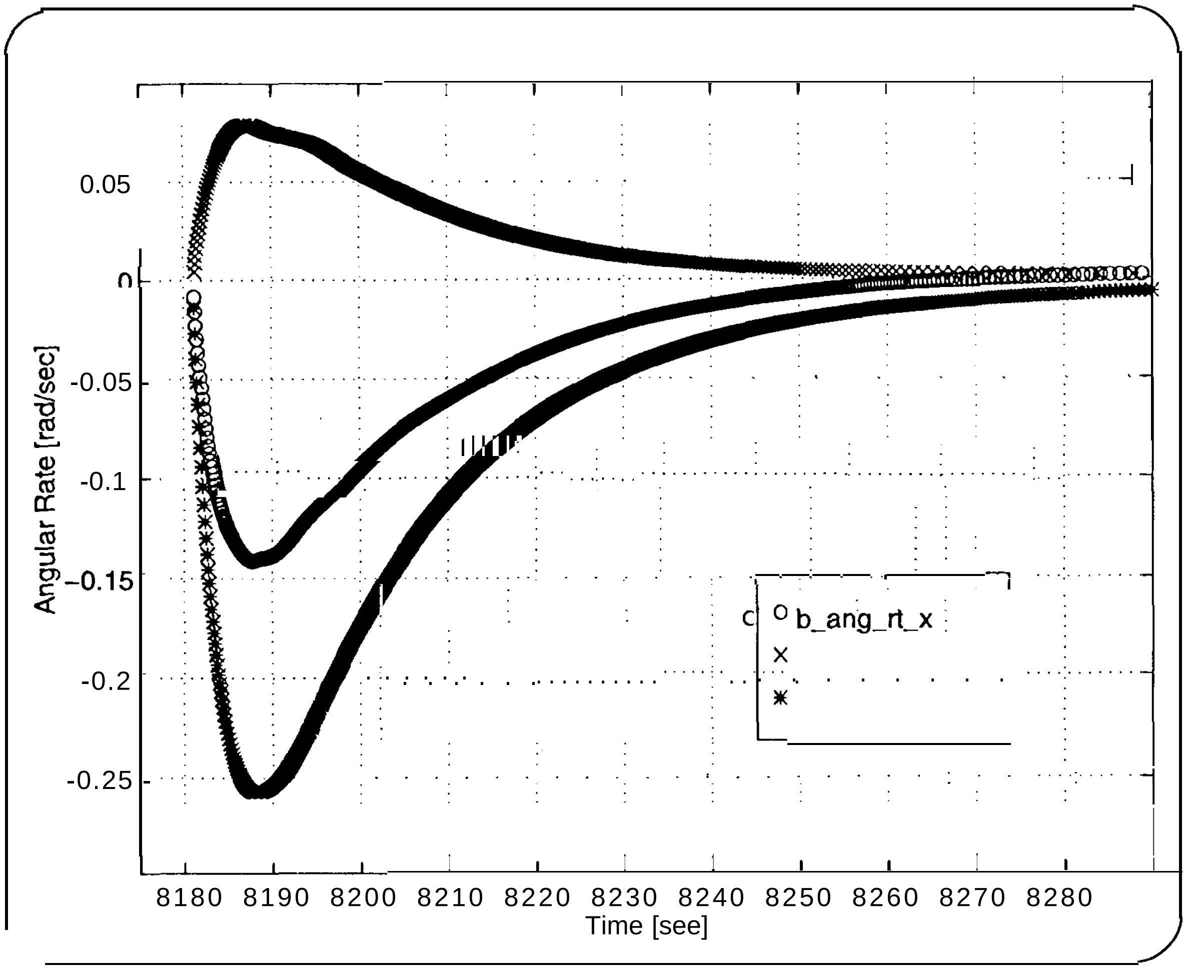


Figure 5-11 Reaction Wheel dynamics model first order anomaly

the integral controller, but not the proportional controller could still be in error if a large, incorrect rate sample was computed. Once the change in time computation was limited, the model did behave as expected with no anomalous error responses.

5.2.3 **Closed Loop Simulation**

A closed loop flow diagram is shown in figure 5.12. This configuration was used to rest the AACS

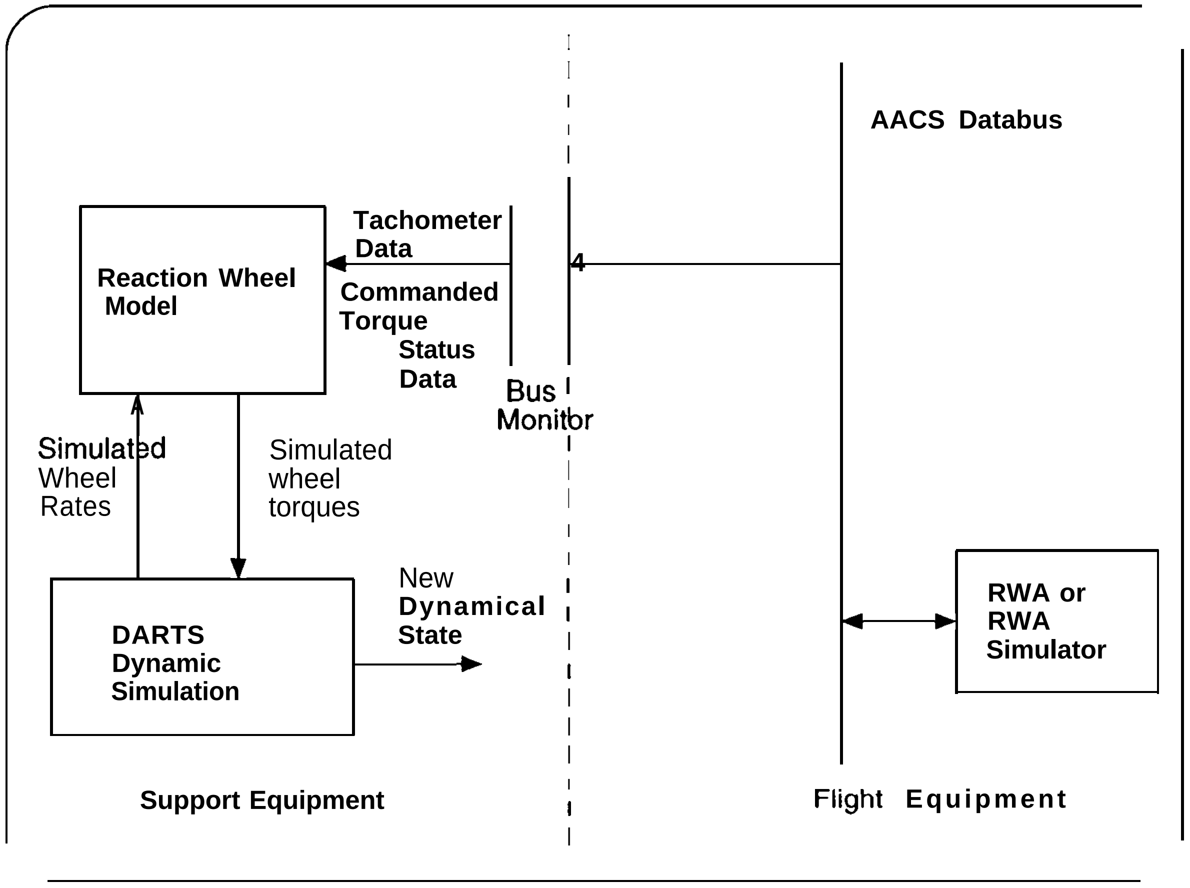


Figure 5-12 Reaction Wheel Simulation closed loop diagram

during the operational modes (OPM) sequence testing. The OPM sequences test the reaction wheels during a series of precision pointing maneuvers simulating science data gathering operations when Cassini performs its orbital tour. The desired profile of the spacecraft angular rate is shown in Figure 5.13. The data was generated during flight software testing on the Flight Software

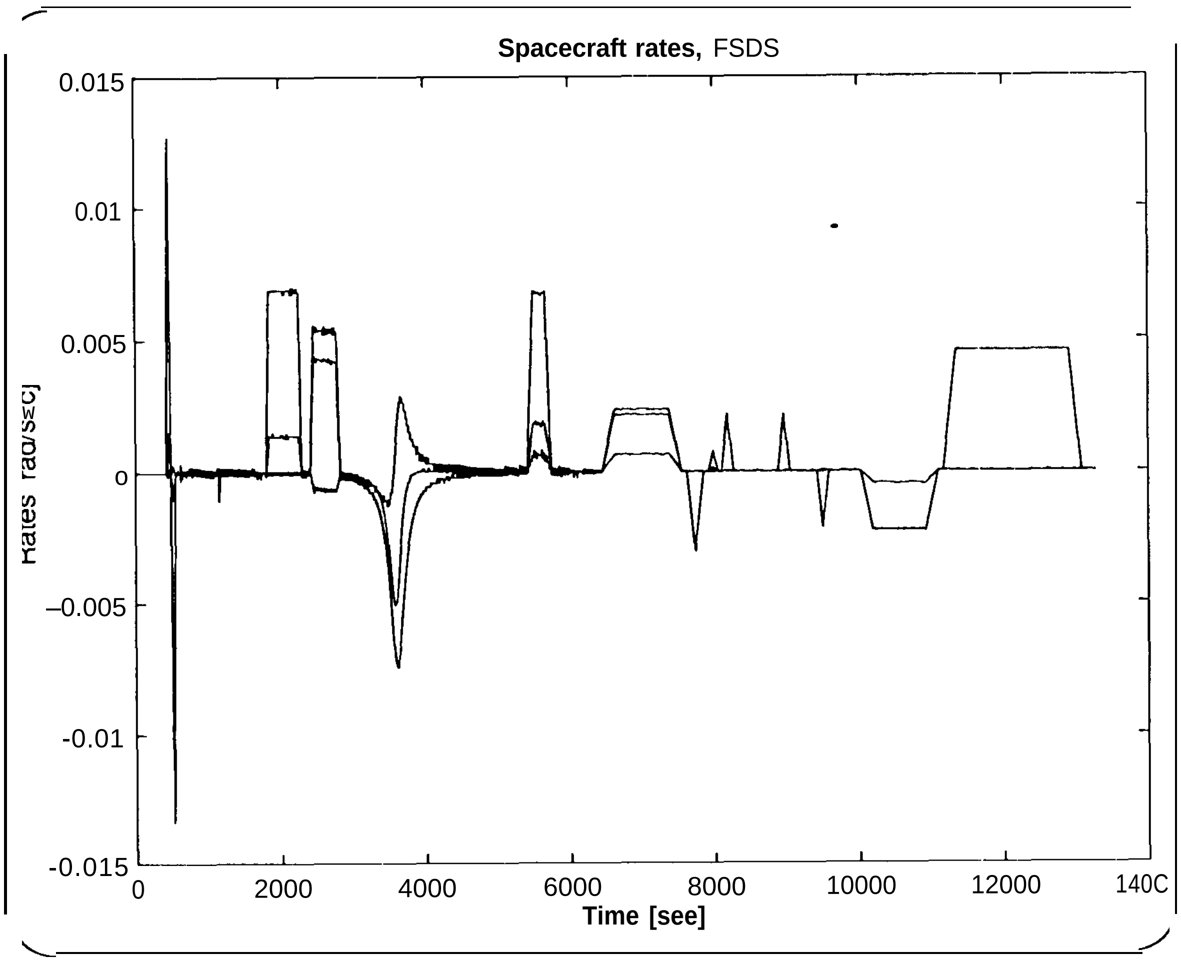


Figure 5-13 RWA OPM Testing- Flight Software Development Station

Development Station (FSDS). The next figure, 5.14, shows the results from the testing in CATS

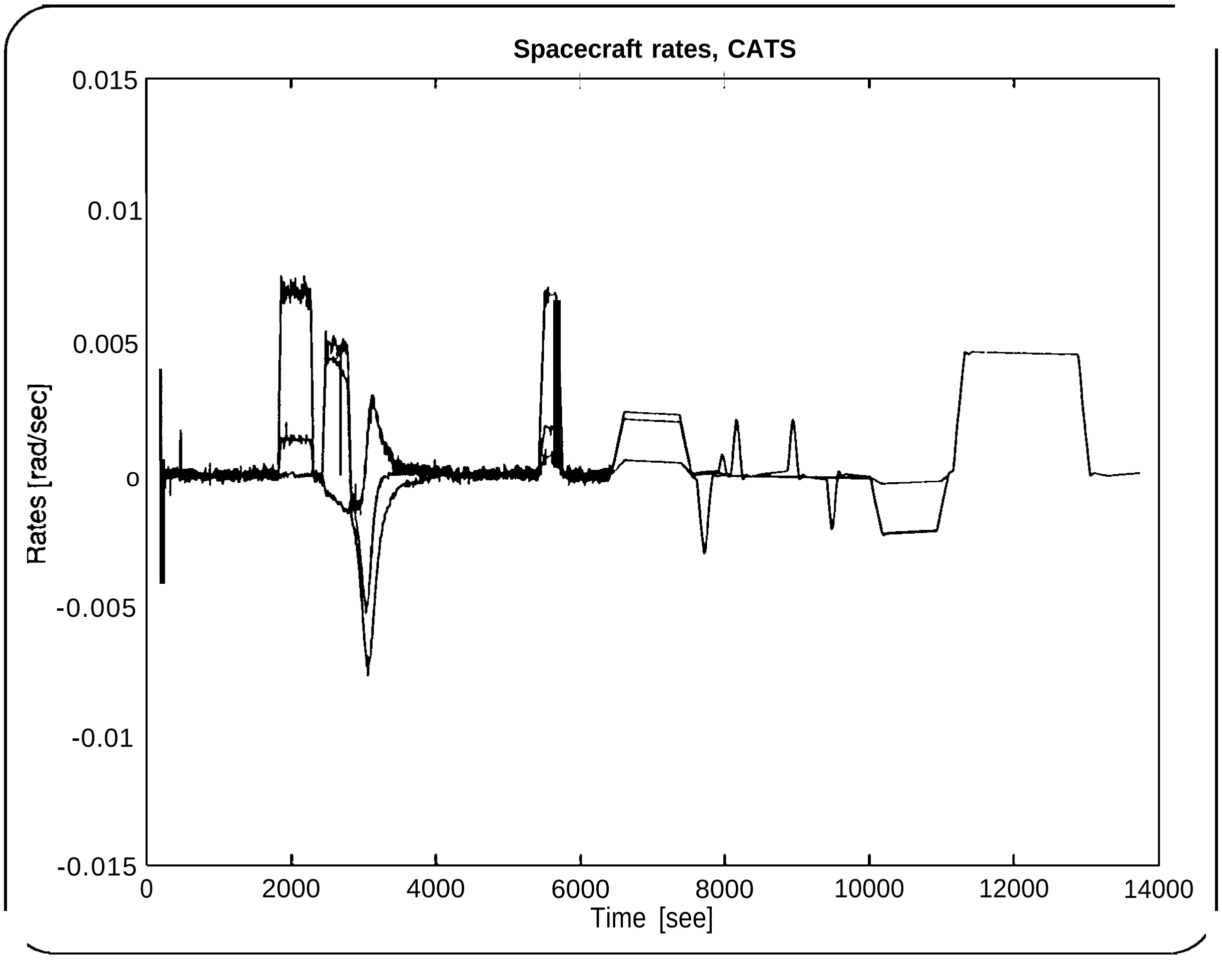


Figure 5-14 RWA OPM Testing- CATS

on GMT day 96-237. The spacecraft is under reaction wheel control for the second part of the sequence. This section can be identified by the reduction in noise of the rate plots. The CATS tests matches very closely with the FSDS run and the closed loop performance of the RWA simulator and the dynamics model was verified.

An important lesson that was learned during the RWA closed loop testing is the value of simulators when software is being tested. As the complexity of software grows, the chance that something will operate incorrect] y grows as well. This is evident from several flight software tests when the software reaction wheel manager was acting incorrectly and was sending torque commands to the reaction wheels periodically. If these commands were sent to the real hardware, the results could have been very unfortunate and the hardware may have been damaged. Thus, the simulators played a critical role in closed loop software testing as well.

5.3 Evaluation

The reaction wheel simulator is the most complicated software simulator due to its internal dynamics and the importance of timing associated with it. Both functionally and in performance, the RWA simulator closely matched the real hardware. The laboratories relied heavily on the simulators during testing and they performed their functions well. Validation activities for the reaction wheels revealed the problem of data corruption within the simulation. This problem was surmounted by implementing filters on incoming reaction wheel data. The fact that the flight and support equipment software do not talk to one another resulted in a difficult implementation of the reaction wheel RTIOU reset response as well. However, even with these problems, the RWA simulator permitted effective testing of the AACS.

Chapter 6

Conclusion

This thesis has investigated the use of simulators during the testing of Cassini's AACS. The simulators have all met their requirements and testing in the three laboratories was improved through the use of these simulators. There are several lessions that have been learned during the testing period.

The first lesson is the importance of clear requirements and objectives in simulation. For example, the fact that the EGA simulators were never meant to simulate a leaded EGA led to the activation of fault protection during ATLO testing. It was the misunderstanding of this fact that led to the fault protection event. An example of clear requirement understanding is the IRU SHARC simulator. It was clear from the beginning that "the support equipment software would not simulated the actual BIT or software download. Since this was specified, testing of these functions was handled elsewhere and this clear expectation led to effective testing.

A second lesson is that there is a tradeoff when deciding on additional command paths in simulators. The Cassini project decided not to add any additional command paths. The advantage of this is that the simulators act just like the hardware in terms of how it interfaces with the support or flight equipment. The disadvantage is that the ability of the tester to inject faults into the system is compromised. The simulators could not be forced to report false information. For example, the only way an IRU simulator could report bad data is if the dynamics simulation reported incorrect data to the IRU. Thus the designer of a simulator must decide between increased ability to inject faults and a simulator that is more like the actual flight equipment.

A final lession that applies not only to these simulators but to the laboratory environment as a whole is that in real time simulation, timing is everything. This was demonstrated with the RWA - RTIOU reset response simualtion as well as the data transfers between the aspects of the simulators. Loss of data bween the RWA simulator and mode] caused testing failures and the biggest problem with the support equipment software as a whole was data loss and corrupriton. Real time simulation is a great asset and an indispensable tool, but care must be taken to preserve data integrity to ensure consistent and reliable testing of subsystem hardware and software.

References

[1] Dahl, P. R. Measurement of Solid State Friction Parameters of Ball Bearings. NASA Document ID 77N33528. 10 March 1977.

[2] Graves, Rick D. Cassini AACS Support Equipment Hardware Design Requirements and Description Document. JPL EM 343-1318, Rev. D. 15 September 1996.

[3] Jain, Abhinandan. DARTS- Dynamics Algorithms for Real-Time Simulation of the CRAF/ Cassini Spacecraft. JPL Document D-9308. January 1992.

[4] Montanez, Leticia M., custodian. AACS Cassini Support Equipment Software Design Requirements and Description Document. JPL IOM 3413-96-122. 12 April 1996.

[5] Rittmuller, Philip A. ITL Testing of the Engine Gimbal Actuators. JPL IOM 3410-95-224 CAS. 15 June 1995.

[6] Walker, W. John, ed. Cassini AACS Interface Control Document. JPL Document D- 12463. Cassini Project Document PD 699-113. Issue #2, 16 June 1995.

[7] Wong, Dr. Edward C., ed. Project Cassini Control Analysis Book. JPL Document D-9638. Cassini Project Document PD 699-410. Update #2,31 January 1994.

Appendix A: Reaction Wheel Databus Transmission Decoding

In section 5.2.1, databus transactions revealed difficulties in simulating the reaction wheel's response to a remote terminal reset. This appendix shows how that data was decoded.

All transmissions on the AACS databus are known as packets. Commands from the AACS Flight Computer are denoted as reaction wheel destination packets since they come the AFC and the destination is the reaction wheel, The designation for this packet is "bm_d_rwx2". This stands for bus monitor destination packet for reaction wheel electronics 2. There are two types of messages that the AFC sends to the reaction wheel-- commands and requests for data. A typical command packet is shown below. In the following explanations, a word is defined as four hexadecimal digits and a byte as 2 hexadecimal digits.

bm_d_rwx2=OOOeae040707   400100400665   CO   96

The first four hexadecimal digits (OOOe) are for support equipment processing and is not used by either the AFC or the RWA. The second "word" consists of two parts. Oxae denotes the address of the destination and stands for RWA2. 04 is a control byte that tells the RWA how to respond to the command. In this instance, 04 simply means that the packet is a command and is coming from the prime AFC. The third word tells the reaction wheel that the source was 07, which is AFC-A, and the number of bytes in this message is 7.

The commands start with OX40. The OX40 is a command to write data to a remote terminal address. Specifically, in this instance, it means to write one byte of data, starting at the address specified by the next byte. So, the 040100 is decoded to mean write one byte of data, starting at address 01, and that data is OXOO. Register 01 of the reaction wheel's remote terminal is the torque enable command, so the AFC is enabling the RWA to accept the torque command.

The next command is similar. It reads4006 65. This means to write one byte of data to the reaction wheel, starting at address 06, and that data is 0x65. This register is the torque command register. It is this register that we observed to fluctuate unexpectedly during the RWA RTIOU reset testing.

The final bytes are CO and 96. The OxcO is a “no operation” command and is needed to expand the packet to the correct length for the databus. The 96 is the checksum of the command,

A packet from the AFC to request data is similar to the command to write data:

bm_d_rwx2 = 0012 ae06070b 00020103000500064004OO 10

The first three words have the same definitions as before. 0012 is used by the support equipment, ae06 says the destination is the RWA2 and the command is from the prime AFC (OX06 has identical meaning to 0x04), and 070b means the source is AFC-A and to expect 11 bytes in this packet (0x0b = 11 decimal).

The first command is OXOO02. This means collect one byte of data from address 02 (read to load RWA tachometer register). 0103 means collect two bytes starting at address 03 (upper and lower bytes of the tachometer). 0005 and 0006 decode to collect one word from registers 5 and 6 respectively (RWA line current and torque command wrap around). Finally, 040400 is a write of one byte to address 04 and that byte of data is OXOO. The Ox 10 is the checksum.

The second type of packet on the databus is from the RWA. This is denoted as bm_s_rwx2, meaning bus monitor source packet from reaction wheel electronics 2. A typical reply is shown below.

bm_s_rwx2 = 00100706 ae09 eOOOOO ff OOOe 9165 CO 76

Again, the first word is for the support equipment, the second denotes the destination as 0x07 (AFC-A) and this time the OX06 is ignored, since the packet is not from the bus controller (AFC-A). ae09 decodes to mean the source is RWA2 and the number of bytes to expect is 9.

The first part of the reply is OxeOOOOO. In binary, this is three 1‘s followed by zeros. The three ones denote this packet as a reply and the zeros indicated no errors in this reply. Next comes the data in the order requested from the AFC. This packet was generated in response to the command to request data that was decoded earlier, so here is the data requested from registers 02 (0xff) ,03 (0x00 Oe) ,05 (0x91) and 06 (0x65). Again, register six contains the torque command wraparound that was observed to behave anomalously during testing. The CO is again the “no operation” command and the 76 is the checksum.

Appendix B: Source Code for software simulations

The following pages include the source code for the following models and simulations:

.Reaction Wheel Assembly Simulator

.Reaction Wheel Dynamics Model

.Engine Gimbal Actuator Dynamics Model

.Inertial Reference Unit Assembly Simulator

.Intertial Reference Unit Dynamics Model

asm_iru.c

```c
/*
 * $Id: asm_iru.c,v 1.5 1997/01/28 20 52 43 itsl Exp $
 *
 */
#include <stdioLib.h>
#include <math.h>
#include "blackboard.h"
#include "vme_addresses.h"
#include "serverDemo.h"
#include "sebbdef.h"
#include "/usr/vw/config/hkv3f/time.h"
#include <rngLib.h>
#include "assembly.h"

#define check_afc   0xffffffff
#define force_write 0x00000000

extern int      poke8()
extern int      peek8()
extern char    *assem_irua;
extern char    *assem_irub;

struct assem_board card[6] = {
        {&assem_irua, 0, 0, 0, 0, 0, 128, 0, 0, 0, -1, 0, 65, 35, 0, 0, 5},
        {&assem_irub, 0, 0, 0, 0, 0, 128, 0, 0, 0, -1, 0, 65, 35, 0, 0, 5},
        {&assem_rwx1, 0, 0, 0, 0, 0, 128, 0, 0, 0, -128, 0, 128, -128, 0, 0, 5},
        {&assem_rwx2, 0, 0, 0, 0, 0, 128, 0, 0, 0, -128, 0, 128, -128, 0, 0, 5},
        {&assem_rwx3, 0, 0, 0, 0, 0, 128, 0, 0, 0, -128, 0, 128, -128, 0, 0, 5},
        {&assem_rwx4, 0, 0, 0, 0, 0, 128, 0, 0, 0, -128, 0, 128, -128, 0, 0, 5}
};

struct assem_board checkpoint card[6] = {
        {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
};
struct assem_board checkpoint2_card[6] = {
        {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
};

/** flag to simulate no download what so ever **/

int             fake_download = 0;
int             iru_noise_on = 1;
int             iru_pwr[2];     /** power status register read from board **/

/** iru scale factors and other "constants" **/

float           iru_bias_sf = 0.0000005;        /** bias scale factor **/
double          iru_angle_sf = 0.00000025;      /** angle scale factor = .25 urad **/
float           iru_timetag_sf = 0.000064;      /** time tag scale factor = 64 usec **/
float           iru_timetag_rollover = 65536.0 * 0.000064;      /** max 16 bit int * sf
= 4.194304 sec **/
double          iru_rollover = 4.194304;        /** largest angle word in rad  **
double          scaled_iru_gyro_sat_limit = 15.0 / 57.29578049044 / 0.0000005
/** saturation limit, 15 degrees/sec in rad/sec and scaled by iru_bias_sf **/
float           iru_cycle_time = 0.01   /** 1 cycle based on 100 hz **/
double          iru_cycle_time_sf = 0.01 * 0.0000005;
/** iru_bias_sf * iru_cycle_time, used to calculate angle in cycle period **/
int             iru_BIT_time = 400;     /** 1 sec, based on 400 Hz **/
int             iru_soft_reset_time = 400 * 100;        /** 100 sec, based on 400 Hz
 **/
double          iru_noise_step = .00000025;     /** rad/cycle **/
double          iru_noise_amp = .000002;        /** rad **/
int             iru_gyrosatr_delay_time = 200 * 100;    /** 200 sec based on 100 Hz
**/

/** generic iru parameters **/

int             iru_datavalid[2];
int             iru_datainvalid = 0x0080;       * iru mode valid, all angles
                                                * invalid */
int             iru_gyrosatr[2];
unsigned int    iru_timetag1[2];
unsigned int    iru_timetag0[2];

/** irua parameters **/
int             gyroA_angle_A0;
int             gyroA_angle_A1;
int             gyroA_angle_A2;
int             gyroA_angle_B0;
int             gyroA_angle_B1;
int             gyroA_angle_B2;
int             gyroA_angle_C0;
int             gyroA_angle_C1;
int             gyroA_angle_C2;
int             gyroA_angle_D0;
int             gyroA_angle_D1;
int             gyroA_angle_D2;

/**  irub parameters **/
int             gyroB_angle_A0;
int             gyroB_angle_A1;
int             gyroB_angle_A2;
int             gyroB_angle_B0;
int             gyroB_angle_B1;
int             gyroB_angle_B2;
int             gyroB_angle_C0;
int             gyroB_angle_C1;
int             gyroB_angle_C2;
int             gyroB_angle_D0;
int             gyroB_angle_D1;
int             gyroB_angle_D2;

/** iru status read sequence status words **/

int             bit_stat[] = {0, 0};
int             bit_crnt[] = {0, 0};
int             bitgyra1[] = {0, 0};
int             bitgyra0[] = {0, 0};
int             bitgyrb1[] = {0, 0};
int             bitgyrb0[] = {0, 0};
int             bitgyrc1[] = {0, 0};
int             bitgyrc0[] = {0, 0};
int             bitgyrd1[] = {0, 0};
int             bitgyrd0[] = {0, 0};
int             bit_cpu1[] = {0, 0};
int             bit_cpu0[] = {0, 0};
int             bit_psm1[] = {0, 0};
int             bit_psm0[] = {0, 0};
int             bit_dnld[] = {0x0, 0x0}
```

```
/** summation angles for a particular gyro **/

int             dyn_iru_rate[2][4];
int             angular_rate[2][4] = {0, 0, 0, 0, 0, 0, 0, 0};
int             dyn_rate_avail[2] = {0, 0};
int             internal_angular_rate[2][4];
int             Eangle[2][4];
double          DEangle[2][4];
int             iru_noise_index[2] = {0, 512};
int             iru_gyrosatr_delay[2][4];
double          iru_noise_array[1024] = {
        -1.72614048501746E-06, 1.25455292857643E-06, 5.83641686687984E-07, 1.27800367515
740E-06,
        1.48657820558932E-07, 1.31909572445318E-06, -3.52305310278122E-07, 1.97763345348
985E-06,
        6.50103664136809E-07, -1.10545356321166E-06, 1.87654872197036E-06, 3.46774260010
270E-08,
        -1.83212247943589E-06, -4.77664359071919E-07, -1.00944783752418E-06, -1.15449740
309575 E-06,
        -4.73929747008347E-07, 1.89942129338022E-06, -8.91091551437990E-07, -4.013854388
73152E-08,
        6.79630415791694E-07, -3.63367265813063E-07, 1.67868110016987E-06,  1 2312015952
4139E-06,
        -5.35845713222801E-07, 5.61465696601252E-07, -1.21746747604819E-06, 9.0161 183643
0167E-08,
        -1.45526869644867E-07, 1.10646012882237E-06, 1.47029802587954E-06,  5.170 S525500
4587E-08,
        1.96625992694314E-06, -7.37128936772115E-07, -2.24485051447476E-09, 1.9935534270
6278E-06,
        -7.54639649174099E-07, -1.81265290577488E-06, 1.07167068517751E-06, -1.727153932
67560E-07,
        -1.92011834470616E-06, -1.07639035396707E-06, 1.10758721735874E-06, -6.54 9436880
91694E-07,
        1.79310004157886E-06, -1.48004444027332E-06, -1.93084101338012E-06, 5.9594334701
3617E-07,
        7.03725322816529E-07, 9.84766742425102E-07, -2.41149046046580E-07,  1.429 3402603
6167E-06,
        -5.39531502721337E-07, -5.79355369052908E-07, 1.81998891117657E-06, 1.0439375258
0646E-06,
        -1.75504180943286E-06, 5.36904124857416E-07, 4.59451538701734E-08,  ..008.9872317
2915E-06,
        2.88532885123957E-07, 4.74796760504963E-07, 7.09007763852420E-07, 5 888 809143466
01E-07,
        1.45884119060671E-06, 6.31722766849071E-07, -1.82535161918240E-07,  1.536 8589841
2868E-06,
        1.62154094216984E-06, -1.08010263792116E-06, 1.78328177877978E-06,  6.2921558560
0968E-07,
        4.17275352453774E-07, -1.67405480833046E-06, -4.49952510210184E-07, 7.7 082175944
6483E-07,
        -1.441048 64333 636E-06, 5.4296 8540261459E-08 ,  4.71921 876477766E-07, 1.81113556398
642E-06,
        1.69315758125426  s-05,  -1.43 2808918 67270E-06, 5.5200668395432 7E-07,  -5.9939492325
444 8E-07,
        1.8521526379300 3E-06,  5.5722  °653352391?-07,  1.0254 8087574458 E- 06, -4.51517  ?57405
592E- 07,
        -2.34773563 315680E-07, 1.9462 8405238477E-06, -5.8073141878 7649E- 07, 2.54364 08925
?2745- 07,
        1.3671667667 3693E-06 , 2.3704 8937867921E-07 ,  -1.3987447 8149150E - 06, -1.4319018727
9782E-06,
        1.78650428 165781E-06,  4.716 909638 85121E-07,  5.82747392627 838E-07,  1.25700574559?
40E- 06,
        8.54 898427733908E-07,  1.15032434074553  E-06, 1.4751909964743 8E-06,  -1.10765149777
967E-07,
        -2.449262105  PL7956E-07,  -1.7 8001776541864E-0 5, 1.50177277611941 E-06 , 7.9179230473
5794E- 07,
        1.461712 88507676E-06,  -1.20067690001 841E-06, 9.0270744 0701975E-07,  4.3516559
9°7524'2S-57,
        1.79 891320830200E - 06,  5.5640865 1384016E-07, 1.772452559155453-05,   -1.6221909
272?7222-0.5,
        8.012491 86549932E - 07,  -5.8044071 8743441E-07 ,  -1.8912422348 7268 E-06, 8.228081
87536834E-07,
        7.8 80575562 795 69E-07,  1.9696037 5823138E- '25,  -4.6261479932 875 0E-07, 1.8 855745
6849962E-06,
        5.2402759 8462745E-07,  -5.53 984881403871E-07,  -1.7537575145  8581E-06,  -1.32074
373287554 E-06,
        1.0195875 8815815E-06,  -6.1911108444 9417E-07,  6.396559 94292808 E-07, 9.1592812
4328318E-07,
        -5.31747644 826551E-07,  -1.3992405351400  8E-06   -4.4422246  0900216E-07, 8.63684
576883537E-07,
        1.40429224484380E-06,  -1.40392070621332E-06,  9.5156160587 8006E-07 , 7.8073943
7291518E-07,
        -1.82797360844401E-06,  1.58241141211951E-06,  1.5000786949 8485E-06, -5.75 8893
66589034E-07,
        -1.97727872653749E-07,  2.64178059684499E-07,  -1.60523  830260729E-06, -1.69055
236597785E-06,
        -1.85103350985967E-06,  6.65598148714414E-07,  1.423 00743816137E-06, -1.995607
92233757E-07.
        2.14327350 821588E-08,  1.0425325205955 8E-06 ,  -1.5794 99747 01612E-06, 1.4388666
6560136 E-07,
        1.48241 872912134E-06,  1.82060613692732E -06,  -1.596  ?12895003455-05,  4.54412??
5798708E-07,
        8.5 888961047041 2E-07, -1.8230 781 64988 84E- 06,   1.1339 8507014246E-06, 1.5 966453
2231196 E-08,
        1.44863691 934359E-07 ,  -1.7932641 1187654E- 07, . 1.24890427394126E- 06, 1.159295
545 03209E-06,
        1.14528558676094E- 06,  6.21508577295680E-07, -4.20801144424049 E- 07, 1.3479553
6614293 E-06,
        1.22    472644408922E-03.009352226373889E-07,  1.97 961404305177E-0 6,  1.1636971
8624144E-06,
        2.31326153061 843 E-07 ,  -5.214 41627310134E-07, -1.48 394572748123 E-06, 1.400387
58846417E-06,
        -5.7549222689923 8E-07, 1.48 804754707333E-06,  -7.7831397 0106793E-07, -5.51968
090760606E-08,
        1.6550 8962989767E-06,  -1.88368337830526E-06 ,  -1.7138 7153 176274E - 07, -1.07417
584 106701E-06,
        1.36010755 281469E-06,  -5.03405909 827214E-07 ,  -7.0 3168002634733E-07 , -1.48722
255533739  s-06,
        -1.79521400779159  E- 06,  2.9904238244 8460E-07 , 8.1762498 1586971E-08 , 1.293 8369
019'99015-05,
        -9.80349833119 223 E-07,  -3.22 807239364525E-07,  -1.4 8514737350629E-06, 1.11732
056 864480E-06,
        -1.794948 80637382E-06 , -1.44579 009232993E-0 5,  4.5 °1035191P7537E-07 , 1.273919
00575841E-06,
        1.29925306700335E-06,  -1.07361880611943E-0 6,  1.83378646097424E - 06, -7.38 7715
87567957E-07,
        8.85240981615573E-07 ,  -7.33554 940417747E-08 , 2.38531422  109772 E-07, 5.9046217
1210287  Z-G7,
        -1.88134803846 805E-06,  -9,912525917  83357E-07,  -1.11831871566813 E-07, 3.91261
441 824921E-07,
        1.4677 8145947215E-06 ,  1.90220 7752259 02E- 07,  -5.515 85309763839E-07, -1.231835
97732152  S-05,
        1.73654273094053  E-07,  -2.74041231740954  E-08,  -8.998408463 58727E-07, 1.033 885
61360226E-06,
        1 95007884509192E- 06, 1.114004737 89626E-06,  1.866 18516004195 - 135,  -1.1281380
065987 0E-06,
        -1.87174805885068 E-06,  -6.01484705836955  E-07, 1.73934313 891949E-06,  -3.79550
983079825E-07,
        -4.41674  890433659E-07, 3.07326013353394  E-07,  6.4 8711717548 872 E-07, -1.919011
13557513 E-07,
        -4.04185057436049 E-07,  1.22  P21355105149E-06,  7.32294732564576 -09,  -1.833754
```

```
42674051E-06,
            6.33016668120087E-07, -1.35756812252294E-06, -1.96992884236677E-07, 1.4925662823
6940E-06,
            -7.92975967995470E-07, -1.36830287383503E-06, 1.61828953177913E-07. 1.3573823725
6769E-06,
            -  70084025716254E-06, -1.21688324897903E-06, 1.85739472240965E-06, -3.754660577
58220E-07
            1 07018786131518E-06, 3.37212161683681E-07, -9.67006652663089E-07, -1.7548392785
4786E-06,
            8 40742858441662E-07, 1.23078415587950E-06, 1.33142462439568E-06. -5.70611762889
878E-07,
            1.34101617353337E-06, 9.50619231056679E-07, -7.33566737474272E-07, -2.8286764226
6914E-07,
            3.24210960586344E-08, -7.00669700322760E-07, 1.19538721345463E-07, 8.33733826245
251E-08
            9.17907934570486E-07, 7.74626653087128E-07, 1.00228745328248E-06, -8.14893860617
658E-08,
            -3.65073927029014E-07, -4.96250343334988E-07, 5.13002158811809E-07, -2.411124588
19950E-08,
            -8.34266740577514E-07, -2.93467448804242E-07, -1.07280624724029E-07. 5.782082447
21453E-07,
            9.93307062445661E-07, -1.53511009478703E-06, 1.21058500579771E-06, 1.72893809561
551E-06,
            1.04515012627326E-06, 1.33507133715104E-06, -1.46583591046011E-06, -1.7809566626
4905E-06,
            9.90641986937650E-08, -6.18797524057346E-07, -3.61463022654510E-07. -1.950684350
98142E-06,
            7.93477218988202E-07, 1.26825006916333E-06, 2.96910569402635E-07, 1.798491738377
31E-06,
            4.99795228975918E-07, -9.18115238034508E-07, 1.09253358484839E-06, 1.76334689504
018E 0.
            1.34353597258157E-06, -7.30675696160605E-07, 1.91324302783199E-06, -1.6172891054
8193E-06,
            -7.52563571639483E-07, 1.17971060874274E-06, 1.11677733047988E-06, -1.8841454811
7660E-06,
            1.62680035704642E-06, 1.63896927526638E-06, 1.71065364232862E-06, 2.513390801014
61E-07,
            2.01190732014196E-06, 1.52618901938237E-06, -1.79083426931540E-06, 8.67213075833
907E-07,
            2,71210433266465E-07, -1.35231018608417E-06, -7.27691870293450E-07, -1.483559437
92243E-06
            1.92253404873192E-06, -1.58796324344946E-06, 1.76257498004955E-08, 1.07022788695
488E-06,
            -9.76706987076563E-07, 1.60652822981845E-06, 7.56368697715501E-07, -3.4055118412
2612E-07
            -6.24544353848896E-07, -1.51409572266773E-06, -1.10852496117387E-06, 9.302330995
54976E-08,
            -9.01457746121162E-07, -9.62245291285206E-07, 1.84560922581640E-06, -8.772491537
25697E-07,
            -9.50431401804257E-07, -4.08335197980622E-07, -6.16812736296086E-07, -4.25814719
550166E-07,
            -1.91444204894465E-06, 1.97991813562623E-06, -1.07680782301499E-06, 3.7527885594
2480E-07,
            1.55897131747282E-07, -1.36559785239195E-07, 2.34756025098133E-08, 2.07179913210
124E-07,
            7.47265821397752E-07, -1.10625825090566E-06, -3.00011791692983E-07, 2.5495411897
2022E-08,
            -1.55475858390734E-06, -1.96669018496587E-06, -1.86938969943497E-06, -1.74298887
756155E-08,
            1.16390113248603E-06, -1.07159613550895E-06, 8.99054969544711E-07, -5.4621835790
6154E-07,
            -1.70469970128539E-06, -1.51411694090945E-07, 1.06491589861929E-06, 1.8771770051
8610E-06,
            -1.79812975979643E-06, -1 37695832173156E-06, 1.89403456814096E-06  -3.982705949
90097E-07,
            1.22210489577834E-06, 7.48998325723439E-07  4.42667696808863E-07  4 33268823
921283E-07,
            -1.14050262489990E-06, 1.58859076971889E-06, -1.15545473548479E-06, 7.292062
92363164E-07,
            1.70595627566081E-07, 2.62179110850390E-07, -1.02851515821760E-06, 1.0664557
5164198E-06,
            1.27886741755277E-06, -1.25041307458337E-07, -1.65496737493874E-06, 1.151922
96962215E-06,
            -1.11843576265108E-06, 1.11165584118435E-06, -7.04225424307576E-07, 1.817125
16308011E-06,
            -1.46980187639019E-07, -2.62148995167013E-08, 1.20300012596829E-06, 6.113321
91442835E-07,
            1.09065286470070E-06, 1.77149079004054E-07, 8.43383321567368E-07, -7.2624642
7516986E-07,
            1.20350698055860E-07, -1.37676297701536E-06, -1.57322943803512E-06, -6.19782
382385085E-07,
            1.28684153035859E-06, -1.63731012182421E-06, 1.63430724153578E-07, -1.230948
59151224E-06,
            1.15563175882727E-06, -1.79535219592781E-06, -4.88250168023807E-07, -1.80447
997129965E-06,
            -1.66654251379057E-06, 6.41586794574924E-07, -1.26674146840152E-06, -1.88764
650131700E-06,
            -6.01972502257613E-08, -2.56732486021394E-07, 8.70006842020257E-08, 5.427937
53458743E-07,
            1.28736719741244E-06, -1.39049202533026E-06, -1.73696161694239E-06, -2.59346
961570393E-07,
            3.00827930462534E-07, -7.86173375423625E-07, 2.30720321622919E-07, 1.0376663
4480763E-06,
            1.14201487815690E-06, 7.22108997750970E-07, 9.22860458306880E-07, 1.23323626
030877E-06,
            -2.22795673730330E-07, 1.46894080606459E-06, -1.59371149650339E-06, 9.072063
99934957E-09,
            -1.69283173565848E-06, 1.52835517798438E-07, 2.59733559469699E-07, -2.273749
93830202E-07,
            1.73327187425316E-06, 1.59865835698079E-06,   38155881380624E-06, 1.59740067
313389E-06,
            1.46214358489897E-06, -1.11748006226971E-06, -9.08097650751263E-07, -3.8144
9714450324E-07,
            -4.99917625260191E-07, 1.34998376279491E-06, 1.87720603154433E-06, -1.465780
70888243E-06,
            4.03749584698629E-07, -6.16730047691802E-07, 1.42762826798333E-07, -1.067627
20549222E-06,
            6.32687386704880E-07, 4.46728758723228E-07, -6.96961899929453E-07, -6.710117
61517749E-07,
            7.74872155612728E-07, 9.72962562283854E-07, -1.83732171727967E-06, -1.654327
14282488E-06,
            5.83752787383290E-07, -1.90212053713252E-06, -1.55128924432370E-06, -1.52722
338226354E-06,
            1.40567795831910E-06, -3.53518365747639E-07, -9.23904191663948E-07, 8.387910
13933658E-07,
            -1.77461264198676E-06, -1.51910031163913E-06,  2 73176339247771E-09, 1.28894
081257211E-06,
            -1.80307714154906E-06, 1.32582093217519E-06  -1.58598754984561E-06  6.623668
12390212E-07,
            1.55374735962188E-06, -4.74061276917449E-07  5 29703328658711E-07  -1.643496
57312262E-06,
            -8.8086274763[illegible]6E-07, -1.76786380314770E-06   6.78859323059671E-07  6 04216
637074743E-07,
            -3.76000950140394E 0.   6 42468745434887E-07, 1.3[illegible]22734[illegible]386[illegible]E-06, 6.687722
96663067E-07,
            -7.49887055866940E-07, -1.[illegible]2392418643944E-06  9 78878579355226E-07, -1.02159
321236666E-08,
            1.25594952001853E-06,  1.73832131131716E-06  4.94140324887771E-07, 1.6714371
848[illegible]462E-06,
            -1.35949884031983E-06, 4.97437221585765E-07, -1.79134631002386E-06, 6.305772
```

```
43226157E-07,
        -2.14711228054440E-07, -1.24809329591630E-06, -1.25675161808999E-06, -3.54604042
315851E-07,
        8.40478143569066E-07, 9.13912119965144E-07, 1.33143708802981E-07, -3.55684391025
057E-07,
        4.75138455110792E-07, -2.26945770238344E-07, 2.91463548138059E-07, 1.00904379959
500E-07,
        -1.83496676953324E-06, -2.05594594719012E-07, -4.45730082737054E-07, -7.63441123
923085E-07,
        -1.77559036646877E-06, 1.70786221990303E-06, 1.37586786623094E-06, -1.7989953854
0878E-06,
        7.26289731865259E-07, 6.65045997532428E-07, -1.46282914363950E-07, 2.98802990440
867E-07,
        9.60695372673308E-07, 8.02326763812930E-07, -1.02316538246635E-06, 9.18609844888
672E-07,
        3.89737202105590E-07, 3.53406159491112E-08, -1.59752308393747E-06, -9.6027971398
5319E-07,
        1.20738521253641E-06, -1.45403748729797E-06, -6.76790915940656E-07, 6.0951162418
4724E-07,
        -2.90876696501548E-07, 8.58405414452460E-07, 7.57921683886424E-07, 1.08170739982
873E-06,
        1.44015600849546E-06, 1.28582940719302E-06, 4.86709732353254E-07, -8.90828106937
535E-07,
        -1.06325225293482E-06, 8.81135037142977E-07, -1.14290076498613E-06, 1.9673706904
8662E-06,
        -1.87917138552850E-06, 7.29539892134058E-07, -1.84956521821339E-06, 1.3722473156
3139E-06,
        1.60880385901128E-06, -1.43118834580052E-06, 1.95675178328972E-06, -6.8499672092
2777E-07,
        -4.60848664521808 4E-07, 5.80208695997240E-07, -2.25713200210761E-07, 4.0713417562
7116E-07,
        -1.47896369145609E-06, 1.32218831975793E-06, -1.80103730370086E-06, -6.295805503
78738E-07,
        -1.60818034095472E-06, -7.20793985165140E-07, 1.64646875654709E-06, -1.168580673
26070E-07,
        -4.97039650344487E-07, -7.01945482905634E-07, -1.18012539389561E-06, 4.526535308
66533E-07,
        -9.68624060119643E-07, 1.23793115038937E-06, 1.18439348163980E-06, 5.34759682875
039E-07,
        6.17780797445982E-07, 3.73007660404395E-07, 1.18267625085984E-06, 1.927693017847
98E-06,
        6.70463104068014E-07, 2.00054586245170E-06, -1.15989298463532E-06, 1.00534879941
447E-06,
        -4.74117116298388E-07, -1.52222621577051E-06, -1.54188288098785E-06, 1.434832063
08230E-07,
        -1.15789233760266E-07, 1.17154532410178E-07, -4.63210711071528E-07, 8.9463672144
6191E-07,
        2.01767488939574E-06, 2.15121487315295E-07, -2.66479424343608E-07, -1.1104343781
0714E-06,
        -4.95918947392729E-07, -1.21757514172826E-06, 1.49653526900442E-06, -8.429009395
14184E-08,
        -1.60120087954993E-06, 6.98206827324566E-07, -2.50953196328130E-07, -5.984686387
49843E-07,
        -1.48843376607317E-06, -1.08775053270868E-06, 8.83576285663518E-07, -6.451915254
74007E-07,
        -1.46970316052249E-06, -8.56000331038225E-07, 1.17277613804125E-06, -1.809605617
74034E-06,
        -1.25273588774907E-06, 9.79680044409568E-07, -4.04065763967160E-07, 1.6884519281
4073E-06,
        1.82551460176787E-06, -7.74206703325946E-07, -4.47009851814399E-07, 1.7943625314
9608E-07,
        1.72150187234368E-06, -1.32853966460580E-06, 1.16213846984722E-06, -1.2618009784
6411E-06,
        1.58776383780222E-06, -4.53693288272893E-07, -1.67862368814318E-06, 8.0083614991
8132E-07,
        -.14459699808163E-07, 1.46443091256841E-06, -1.46174983372908E-06, 1.902579
03494556E-06,
        -1.68571499530913E-06, -3.81830020325011E-08, 1.33699748182793E-06, -8.44717
701912642E-07,
        1.61756992828305E-06, -1.08206161469241E-06, -4.91843153403989E-07, -7.0316
6825284522E-07,
        1.07056178446277E-06, -1.10663262256200E-06, 1.404906202974 40E-06, -7.025 665
02786163E-07,
        6.58383859650884E-07, 1.91542091802197E-06, 1.50418550788638E-06, -9.1312548
2380409E-07,
        -6.52705699363432E-07, -1.50350903713117E-06, 1.39386894847744E-06, 8.019635
80183545E-07,
        -7.20141287484543E-07, 1.65705837600516E-06, -6.34930504998460E-07, 1.890444
50886153E-06,
        -8.14114560987376E-07, 1.56581573541117E-06, -1.61854635294069E-06, 3.788893
09202848E-07,
        -2.9986057006992 0E-07, 2.43416298141286E-10, 3.60726066128746E-07, 1.8652938
7823035E-06,
        7.95087643408242E-07, 1.97483508774822E-07, 7.12057027233185E-07, 2.8041730
9337835E-07,
        -1.79503402206054E-06, -2.06806416849552E-08, -1.35030345469988E-06, -1.3312
9618688734E-06,
        6.73796879731924E-07, -1.82106055911681E-06, -9.452327266597 06E-07, -1.18911
055704028E-06,
        1.92785880678923E-06, -1.57330024327045E-06, -1.55614324897761E-06, 3.214677
02757880E-07,
        4.76021985514395E-07, -1.23865528214849E-06, 1.10960714761755E-06, -1.289909
37183772E-06,
        -2.41380659539530E-07, -4.81982581063580E-07, 6.72661487572 85 6E-07, 9.401925
60351989E-08,
        -6.69302059740337E-07, -3.91200371655190E-07, 2.00654711858899E-06, -1.89708
817847951E-06,
        1.93136479366219E-07, -1.88871972421205E-06, 3.59231984838513E-08, -7.564681
42222523E-07,
        -1.48926443499726E-06, -1.67587809921606E-06, 4.08208387530282E-07, -1.43945
640358195E-06,
        9.25084105043329E-07, -9.53506207500340E-07, -8.84458105505894E-07, -7.52488
952863394E-07,
        -4.13463025067455E-08, -7.45479166507320E-07, -6.71831282673087E-07, -8.1681
9201634106E-07,
        1.97921531746589E-06, 1.04476831965702E-06, -8.24382932455578E-07, -1.85338 9
05721396E-06,
        -6.71129511966510E-07, -1.00743108415414E-06, -2.40130025358929E-07, 5.51720
835150062E-07,
        1.75615947183273E-06, 8.62279463548106E-07, -9.10339809439195E-07, -1.743429
37458256E-06,
        1.07249231469907E-06, 1.47721850568667E-06, 1.60189060895426E-06, 1.41169863
991885 E-06,
        -1.68607865729148E-07, -7.40197348114409E-07 1.99184912480257E-06, 1.282219
96753476E-06,
        9.26990395275598E-07, -1.18967779001182E-07, -3.61412128677822E-07, -1.44116
393036295E-06,
        1.45781990511188E-06, -8.60182438454411E-07, 1.97310154460805E-06, 1.4850487
2758158E-06,
        -4.60220394994227E-07, -1.28558902029945E-06 -4.20180749107645E-07, -1.3206
1192635322E-06,
        1.94210058447284E-06, -3.90527181283109E-07, 1.29752563369599E-07, -3.740128
08818095E-07,
        -1.18660865092910E-06, 1.42729903710245E-06, -1.77781104255461E-06, 1.747776
46278297E-06,
        -1.31835472675930E-06, 2.00237980829929E-06, -1.36371771526075E-07, -1.31651
012317864E-06,
        -1.08569657449232E-06, -6.21361902508948E-07 -1.67334598410227E-06, -1.5243
8477690154E-06,
        -7.04901123382831E-07, 9.08459988892244E-07, 1.22753272701588E-06, 1.9285121
```

```
6625202E -06,
        1.68298737034929E-06, -.435216943891 19E-07,  1.4632366286874 8E-06, -4.7366320245
7713E-07,
        -.31858091957727E-06,  1.8704176632842 4E-06,  -1.32282243460977E-06,  4.5793377757
8608E-07,
        3.578900907412 09ɛ-09,  -.258253331 54108E-06,  1.15970188358645E-06,  8.67376187677
122E-08,
        -9.23044782644040E-07, 3.60215260412862E-08,  -.37392053013337E-06,  -1.328854848
64026E-06,
        7.91096792300731E-07,  6.30840194361716E-07,   6.72098432722892E-07,  -8.2659420083
4804E-07,
        7.28832487306107E-07, -9.82851106351550E-07,  -7.99744669491115E-07,  -1.718573392
99621E-06,
        -1.01649812863852E-06,  2.87310248844871E-07,  1.34965662990357E-06,  6.83844833383
975E-07,
        1.64888941743967E-07, -1.06395933805357E-06,  -1.01928250761673E-06,  -1.617180412
42984E-06,
        -1.82577402793792E-06, 1.81456722105755E-06,  -8.96394968001713E-07,  -¿„666450050
19182E-06,
        1.51237389288096E-06, 8.34966177163934E-07,   8.51285468057094E-07, -1.2245352645
7936E-06,
        -2.97580091969297E-07  1.58989846980370E-06, 7.20525440434163E-07, 5.37023701735
617E-08,
        1.18793851485636E-06, -1.15997572644853E-06, -2.75399168814106E-08, -1.066015384
16483E-06,
        6.57939561821602E-07, -1.85445999474637E-06, 2.92112411367414E-07, 1.95244705879
701E-06,
        -5.53849618257518E-07  8.32719336362474E-07, 6.45914180976528E-07, -1.7139459967
1977E-06,
        1.63781453070333E-06, -1.32203153787142E-06, 7.01493577079345E-07, 2.00910212255
485E-06,
        7.32379549349014E-07, -4.76995944243614E-07, 4.98748951847049E-07, 1.02669679268
771E-06,
        1.76476093563047E-06, 1.05981623184704E-06,  1.86215759410794E-06, -6.0474205331
0253E-07,
        1.52918090927409E-06, 2.02507072934970E-06,  1.11712498495434E-06, 1.22936292913
646E-06,
        -1.65116385144453E-06, 9.79639475428436E-07, 1.73624580913446E-06, -4.8748328094
4583E-07,
        -6.53494584305905E-08, -4.66080327606589E-07  1.10439103672808E-06, -1.833235429
88791E-06,
        -1.43316798527602E-06, -1.39484194236342E-06, -3.68873270533372E-07, -1.32630876
909036E-06,
        9.22651883032789E-07, 5.35140322158349E-07, -5.05828742127814E-08, -9.3568477047
5522E-07,
        -1.43633491831141E-06, 1.12302760585641E-06, -5.19314379427760E-07, -4.038111050
14287E-07,
        9.04417741668781E-07, 7.09458799488421E-07, 1.70444963963279E-06, -1.53915599055
723E-07,
        8.14839276696371E-07, -5.97545535962650E-07, -1.45544314158455E-06, -1.526081856
33588E-07,
        -1.59806756557532E-07, -1.50549530136671E-06, -1.56876235039886E-06, -4.65418828
053791E-07,
        -1.24811624762645E-07, 1.00504690361566E-06, -1.58805666439897E-06, 1.7632728230
6992E-06,
        3.49501448412299E-07, -1.83901436172665E-06, -1.16272768173583E-06, -1.908407725
85761E-06,
        -1.94703141932331E-08, -3.05195908591801E-07, -2.55128270428111E-07, -4.02562333
391336E-07,
        3.45615302891836E-07, -1.68546623894339E-06, -1 56878067551215E-06, 1.4806151788
3704E-06,
        -1.54890835279371E-06, 1.67815309476442E-06, -7 40009306065688E-07, -2.384736337
71709E-07,
        -1.37720680395068E-06, -3.31400748881971E-07, 5 84702072221284E-07, 7.3752928969
6196E-07,
        1.8770267603337 9E-07, -4.02924533681891E-07, -5.06436459881842E-07, -1.17277
425765459E-06,
        1.42607857425684E-06, -1261366812 88989ɛ-06, 1.45417259802680E-06, 1.0512315
0642828E-06,
        -3.23414959419556E-07,  -1.22651796791827E-06′  -1.10197903125281E-06, 2.01363
249465723E-07,
        5.81297032159231E-08,  9.83479134684587E-07,  4.35468823102332E-08,  -1.0366832
2992400E-06,
        2.02684591882743E-06, -1 13097268119815E-06,  4.69764736722627E-07,  7.12672 86
5978092E-07,
        1.12569016646982E-06, -1.50414253719575E-06,  -1.07594070771847E-06, -3.21187
810678506E-07,
        -1.38088084062484E-06,  9.77630543854342E-07,  -9.62461248082881 E-07, -¿„89560
330843495E-07,
        4.30292321729281E-07, -7.07553126529290E-07, -2.877524794921 51E-07, 1.105223
7755375 9ɛ-06,
        -1.40232199615012E-06,  -3.78649004170862E-07,  -6.42061724566750E-07,  -¿„1627
6510507123 ¿-07,
        1.64815947210300E-06,  5.612807872422 58E-07,  1.59772444077195E-06,  -1.9012027
1087721E -06,
        3.92708405388783E-07,   1.08715169937486E-06, 1.30308556158586E-06, 1.1053857
5884286E-06,
        -7.91386229732924E-07,  -1.62896863193113E-06,  4.34687046472068E-07,  1.7952 60
29842235E-07,
        -1.40445270562428E-06, 8.29781692223652E-07, 1.75740965621895E-06, -3.090847
55431896E-08,
        -1.50018740010604E-06,  -5.6560530855982 0ɛ-07,  1.55321503377717E-06, 1.280138
85216046E-06,
        -1.16566840536745E-06, -6.52324123446823E-07, 9.80963051411330E-07, -1.36364
205120807E-06,
        1.35111768760645E-06, 4.17069704686053E-07, 3.37960240478444E-07, -1.6390359
0140774E-06,
        -7.08578411960828E-07, 1.84319131184423E-06, 1.40235552829100E-06, 6.3986282
5329641E-07,
        8.56056441794058E-07, -1.05615044206538E-06, -1.05546262172385E-09, -9.42175
545696473E-08,
        -1.16703021744430E-06, 5.21486958743170E-07, -1.37957316576796E-06, 1.907474
88838900E-06,
        1.23700213083418E-06, 1.39129696101873E-06, -1.45605521342072E-06, 1.0765038
1511118E-07,
        -2.83189277831623E-07, 1.92021253057288E-06, -1.15827200238624E-06, -8.64929
011299081E-07,
        -9.61834281333590E-07, 1.01229290477067E-06, -1.56280391559527E-06, 1.390197
29641773E-06,
        1.37665010175446E-06, 5.82839537942762E-07, -1.25656518770267E-06, 1.0639887
8798966E-06,
        -5.70231703408807E-07, -2.13705100976778E-08, 1.61436402418532E-06, -9.45697
503710589E-07,
        2.31518659457327E-07, 3.92110983677179E-07, 1.31749135149351E-06, 1.85439445
508457E-06,
        8.65547790091559E-07, 1.64296466753123E-06, -5.65285264565559E-07, 1.7202041
7134401E-06,
        -2.80581340372115E-07, 4.29936856806205E-07, 1.44864274400681E-06, -1.139177
36536217E-06,
        -1.53166161544967E-06, 1.31240798191820E-06, -1.31921363812076E-06, -2.63342
594037272E-08,
        -1.94291170779201E-06, 1.19425186236904E-06, -1.77864935302958E-06, 1.322157
80497720E-06,
        -3.49861255489350E-07, -2.62355573995900E-08, -1.23545034469329E-07, 1.70192
107799609E-06,
        5.07116624896865E-07, -1.83125979343661E-06, 2.19209774181924E-07, 1.6669289
9519482E-06,
        1.64263897576920E-06, 6.61066397618357E-07, 1.50650152921614E-06, 4.25638626
775617E-07,
        1.52521292034018E-06, 1.97863920832445E-06, -6.00685358798245E-07, -1.287232
```

```
48481058E-06,
-7.63119517406648E-07,-1.38703519677052E-06, 1.28065404916377E-06,  -9.32724665427880E-0
8);


/**  Internal iru model data **/

int            last_iru_pwr[2] = {0, 0};
int            iru_mode[2] = {0, 0};   /**  0=power off,
                                            1=power turned on,
                                            2= download,
                                            3= software reset,
                                            4=normal**/
int            iru_timer[2] = {0, 0};
int            gyro_sat_mask[4] = {0x20, 0x10, 0x08, 0x04};
float          iru_timetag[2] = {0.0, 0.0};
int            iru_1024_hz_cycle = 0;
int            iru_channel = 0;

/**  iru cycle execution array  **/

/**  In this array, the  IRU SIM  will be executed if the value is not zero. **/
/**  The ms cycle number was used, as opposed to just "1", so that the reader **/
/**  will be able to easily located the desired slot number.  Note:  do not **/
/**  use zero for slot zero because that will indicated not to be use the slot. **/
/**  If zero needs to be used, cheat by inserting a "1". **/
/**  This array is set up so that the iru is executed on the cycle before **/
/**  the cycle where the data needs to be available.  In this manner the **/
/**  data will be available when the flight system accesses it.  That of course **/
/**  assumed that the flight system and the sim are synchronized which is **/
/**  probably not the situation which then means that one does not care. **/
/**  This scheme **/
/**  was adopted because at 400 Hz there is not an integral number that **/
/**  divides into 32768, the clock frequency.  As a result any regular **/
/**  update will begin to slide with respect to the rest of the system. **/

int            iru_cycle_array[1024] = {
        0, 1, 0, 0, 4, 0, 6, 0, 0, 9, 0, 11, 0, 0, 14, 0,
        16, 0, 0, 19, 0, 0, 22, 0, 24, 0, 0, 27, 0, 29, 0, 0,
        32, 0, 34, 0, 0, 37, 0, 39, 0, 0, 42, 0, 0, 45, 0, 47,
        0, 0, 50, 0, 52, 0, 0, 55, 0, 57, 0, 0, 60, 0, 0, 63,
        0, 65, 0, 0, 68, 0, 70, 0, 0, 73, 0, 75, 0, 0, 78, 0,
        80, 0, 0, 83, 0, 0, 86, 0, 88, 0, 0, 91, 0, 93, 0, 0,
        96, 0, 98, 0, 0, 101, 0, 103, 0, 0, 106, 0, 0, 109, 0, 111,
        0, 0, 114, 0, 116, 0, 0, 119, 0, 121, 0, 0, 124, 0, 0, 127,
        0, 129, 0, 0, 132, 0, 134, 0, 0, 137, 0, 139, 0, 0, 142, 0,
        144, 0, 0, 147, 0, 0,150, 0, 152, 0, 0, 155, 0, 157, 0, 0,
        160, 0, 162, 0, 0, 165, 0, 167, 0, 0, 170, 0, 0, 173, 0, 175,
        0, 0, 178, 0, 180, 0, 0, 183, 0, 185, 0, 0, 188, 0, 0, 191,
        0, 193, 0, 0, 196, 0, 198, 0, 0, 201, 0, 203, 0, 0, 206, 0,
        208, 0, 0, 211, 0, 0, Z1S, 0, 216, 0, 0, 219, 0, 221, 0, 0,
        224, 0, 226, 0, 0, 229, 0, 231, 0, 0, 234, 0, 0, 237, 0, 239,
        0, 0, 242, 0, 244, 0, 0, 247, 0, 249, 0, 0, 252, 0, 0, 255,
        0, 257, 0, 0, 260, 0, 262, 0, 0, 265, 0, 267, 0, 0, 270, 0,
        272, 0, 0, 275, 0, 0, 278, 0, 280, 0, 0, 283, 0, 285, 0, 0,
        288, 0, 290, 0, 0, 293, 0, Z9S, 0, 0, Z9e, 0, 0, 301, 0, 303,
        0, 0, 306, 0, 308, 0, 0, 311, 0, 313, 0, 0, 316, 0, 0, 319,
        0, 321, 0, 0, 324, 0, 326, 0, 0, 329, 0, 3 31, 0, 0, 334, 0,
        336, 0, 0, 339, 0, 0, 342, 0, 344, 0, 0, 347, 0, 349, 0, 0,
        352, 0, 354, 0, 0, 357, 0, 359, 0, 0, 362, 0, 0, 365, 0, 367,
        0, 0, 370, 0, 372, 0, 0, 375, 0, 377, 0, 0, 380, 0, 0, 383,
        0, 385, 0, 0, 388, 0, 390, 0, 0, 393, 0, 395, 0, 0, 398, 0,
        400, 0, 0, 403, 0, 0, 406, 0, 408, 0, 0, 411, 0, 413, 0, 0,
        416, 0, 418, 0, 0, 421, 0, 423, 0, 0, 426, 0, 0, 429, 0, 431,
        0, 0, 434, 0, 436, 0, 0, 439, 0, 441, 0, 0, 444, 0, 0, 447,
        0, 449, 0, 0, 452, 0, 454, 0, 0, 457, 0, 459, 0, 0, 462, 0,
        464, 0, 0, 467, 0, 0, 470, 0, 472, 0, 0, 475, 0, 477, 0, 0,
        480, 0, 482, 0, 0, 485, 0, 487, 0, 0, 490, 0, 0, 493, 0, 495,
        0, 0, 498, 0, 500, 0, 0, 503, 0, S0S, 0, 0, 508, 0, 0, 511,
        0, 513, 0, 0, 516, 0, 518, 0, 0, 521, 0, 523, 0, 0, 526, 0,
        528, 0, 0, S 31, 0, 0, 534, 0, 536, 0, 0, 539, 0, 541, 0, 0,
        544, 0, 546, 0, 0, 549, 0, SS¿, 0, 0, 554, 0, 0, 557, 0, 559,
        0, 0, 562, 0, 564, 0, 0, 567, 0, S69, 0, 0, 572, 0, 0, 575,
        0, 577, 0, 0, 580, 0, 582, 0, 0, 585, 0, 587, 0, 0, 590, 0,
        S9Z, 0, 0, S9S, 0 0, S98, 0, 600, 0, 0, 603, 0, 605, 0, 0,
        608, 0, 610, 0, 0 613, 0, 615, 0, 0, 618, 0, 0, 621, 0, 623,
        0, 0, 626, 0, 628 0, 0, 631, 0, 633, 0, 0, 636, 0, 0, 639,
        0, 641, 0, 0, 644 0, 646, 0, 0, 649, 0, 651, 0, 0, 654, 0,
        6S6, 0, 0, 659, 0, 0, 66Z, 0, 66L, 0, 0, 667, 0, 669, 0, 0,
        672, 0, 674, 0, 0, 677, 0, 679, 0, 0, 682, 0, 0, 685, 0, 687,
        0, 0, 690, 0, 69Z 0, 0, 69S, 0, 697, 0, 0, 700, 0, 0, 703,
        0, 7 05, 0, 0, 708, 0, 7IO, 0, 0, 713, 0, 715, 0, 0, 718, 0,
        7ZO, 0, 0, 723, 0, 0, 726, 0, 7 28, 0, 0, 731, 0, 733, 0, 0,
        7 36, 0, 7 38, 0, 0, 7 41, 0, 743, 0, 0, 746, 0, 0, 749, 0, 751,
        0, 0, 754, 0, 756 0, 0, 759, 0, 761, 0, 0, 764, 0, 0, 767,
        0, 769, 0, 0, 77Z, 0, 774, 0, 0, 777, 0, 779, 0, 0, 782, 0,
        7 84, 0, 0, 787, 0 0, 790, 0, 79Z, 0, 0, 795, 0, 797, 0, 0,
        800, 0, 802, 0, 0 805, 0, 807, 0, 0, 810, 0, 0, 813, 0, 815,
        0, 0, 818, 0, ƎZO, 0, 0, 823, 0, 825, 0, 0, 828, 0, 0, 831,
        0, 833, 0, 0, 836 0, 838, 0, 0, 841, 0, 843, 0, 0, 846, 0,
        ¿96, 0, 0, 851, 0 0, 854, 0, 856, 0, 0, 859, 0, 861, 0, 0,
        864, 0, 866, 0, 0 869, 0, 871, 0, 0, 874, 0, 0, 877, 0, 879,
        0, 0, 882, 0, 884 0, 0, 887, 0, 889, 0, 0, 892, 0, 0, 895,
        0, 897, 0, 0, 900 0, 902, 0, 0, 905, 0, 907, 0, 0, 910, 0,
        912, 0, 0, 915, 0 0, 918, 0, 920, 0, 0, 923, 0, 925, 0, 0,
        928, 0, 930, 0, 0 933, 0, 935, 0, 0, 938, 0, 0, 941, 0, 943,
        0, 0, 946, 0, 948 0, 0, 951, 0, 953, 0, 0, 956, 0, 0, 959,
        0, 961, 0, 0, 964 0, 966, 0, 0, 969, 0, 971, 0, 0, 974, 0,
        976, 0, 0, 979, 0, 0, 982, 0, 984, 0, 0, 987, 0, 989, 0, 0,
        992, 0, 994, 0, 0, 997, 0, 999, 0, 0, 1002, 0, 0, 1005, 0, 1007,
0, 0, 1010, 0 1012, 0 0, 1015, 0, 1017, 0, 0, 1020, 0, 0, 1023};

/** Test visability data for debugging purposes **/

int            Jim_gyr_stra = 0;
int            Jim_gyr_strb = 0;
int            trap_angular_rate_a;
int            trap_angular_rate_b;
int            trap_angular_rate_c;
int            trap_angular_rate_d;
int            trap_i;
int            trap_mask;
double         trap_angle_rate;
int            soft_reset_flag[2] = {0, 0};
int            dwn_ld_done[2] = {0, 0};
int            FuncCodeA;
int            FuncCodeB;
int            MspaceA;
int            MspaceB;

/** unused parameters, can these be deleted??  **/

RING_ID        iru_ring_Q = 0;
int            iru_task_id = 0;
int            iru_Q_task();

/**  iru model code  **/

iru_reset(iru_device_id)
       int            iru_device_id;  /* irua = 0, irub = 1 */
```

```
{

        **
        *  IRU will reset as a result of RTIOU reset or input power falling
        *  below minimum levels (except transients).  Following a reset
        *  code and data must be downloaded to the IRU and gyro angle
        *  counters are cleared.
        **/

       if (iru_device_id == 0) {

               gyroA_angle_A0 = 0.
               gyroA_angle_A1 = 0.
               gyroA_angle_A2 = 0.
               gyroA_angle_B0 = 0.
               gyroA_angle_B1 = 0.
               gyroA_angle_B2 = 0.
               gyroA_angle_C0 = 0.
               gyroA_angle_C1 = 0.
               gyroA_angle_C2 = 0.
               gyroA_angle_D0 = 0.
               gyroA_angle_D1 = 0.
               gyroA_angle_D2 = 0.
       }

       if (iru_device_id == 1

               gyroB_angle_A0 = 0
               gyroB_angle_A1 = 0
               gyroB_angle_A2 = 0
               gyroB_angle_B0 = 0
               gyroB_angle_B1 = 0
               gyroB_angle_B2 = 0
               gyroB_angle_C0 = 0
               gyroB_angle_C1 = 0
               gyroB_angle_C2 = 0
               gyroB_angle_D0 = 0
               gyroB_angle_D1 = 0
               gyroB_angle_D2 = 0




       };
       iru_datavalid[iru_device_id] = 0x80;
       iru_gyrosatr[iru_device_id] = 0x00;
       iru_timetag0[iru_device_id] = 0x00;
       iru_timetag1[iru_device_id] = 0x00;
       iru_timetag[iru_device_id] = 0; /* zero out time ctr *
       Eangle[iru_device_id][0] = 0.
       Eangle[iru_device_id][1] = 0.
       Eangle[iru_device_id][2] = 0.
       Eangle[iru_device_id][3] = 0.
       DEangle[iru_device_id][0] = 0;
       DEangle[iru_device_id][1] = 0;
       DEangle[iru_device_id][2] = 0;
       DEangle[iru_device_id][3] = 0;
       iru_gyrosatr_delay[iru_device_id][0] = 0.
       iru_gyrosatr_delay[iru_device_id][1] = 0.
       iru_gyrosatr_delay[iru_device_id][2] = 0.
       iru_gyrosatr_delay[iru_device_id][3] = 0
```

```
iru_rtiou_write(id, out, afc)
        int             id;     /**coorsponds to iru device, irua=0 and irub=1 **/
        int             out;    /** 16=time and status, 8=chan A, 4=chan B, 2=chan C

                                        1=chan D, 31=all **/
        int             afc;    ** 0xffffffff=check afc read counter,
                                        0x00000000=ignore afc interaction **/



        if ((id == 0) &&  (assem_irua_write_ok == 1)    (afc == 0))) {

                *
                * Ok to write to assembly unit, if the flight computer
                * starts a transaction while in this block, we should be
                * able to finish before the assembly unit actually responds
                * to the read but we will protect ourselves by marking the
                * data invalid during the write process
                */

                /* poke8(assem_irua + (0x10 << 1  + 1, iru_datainvalid);  */


                if ((out & 0x0010) != 0) {
                        poke8(assem_irua + (0x11 << 1) + 1, iru_gyrosatr[0]),
                        poke8(assem_irua + (0x1e << 1) + 1, iru_timetag1[id]),
                        poke8(assem_irua + (0x1f << 1) + 1, iru_timetag0[id]),
                        poke8(assem_irua + (0x20 << 1) + 1, bit_stat[id]);
                        poke8(assem_irua + (0x21 << 1) + 1, bit_crnt[id]);
                        poke8(assem_irua + (0x2a << 1) + 1, bit_cpu1[id]);
                        poke8(assem_irua + (0x2b << 1) + 1, bit_cpu0[id]);
                        poke8(assem_irua + (0x2c << 1) + 1, bit_psm1[id]);
                        poke8(assem_irua + (0x2d << 1) + 1, bit_psm0[id]);
                        poke8(assem_irua + (0x2e << 1) + 1, bit_dnld[id]);
                };
                if  (out & 0x0008) != 0) {
                        poke8(assem_irua + (0x12 << 1) + 1, gyroA_angle_A2);
                        poke8(assem_irua + (0x13 << 1  + 1, gyroA_angle_A1);
                        poke8(assem_irua + (0x14 << 1) + 1, gyroA_angle_A0);
                        poke8(assem_irua + (0x22 << 1) + 1, bitgyra1[id]);
                        poke8(assem_irua + (0x23 << 1) + 1, bitgyra0[id]);
                };
                if ((out & 0x0004) != 0) {
                        poke8(assem_irua + (0x15 << 1) + 1, gyroA_angle_B2)
                        poke8(assem_irua + (0x16 << 1) + 1, gyroA_angle_B1)
                        poke8(assem_irua + (0x17 << 1) + 1, gyroA_angle_B0)
                        poke8(assem_irua + (0x24 << 1) + 1, bitgyrb1[id]);
                        poke8(assem_irua + (0x25 << 1) + 1, bitgyrb0[id]);
                };
                if ((out & 0x0002) != 0) {
                        poke8(assem_irua + (0x18 << 1) + 1, gyroA_angle_C2),
                        poke8(assem_irua + (0x19 << 1) + 1, gyroA_angle_C1),
                        poke8(assem_irua + (0x1a << 1) + 1, gyroA_angle_C0),
                        poke8(assem_irua + (0x26 << 1) + 1, bitgyrc1[id]);
                        poke8(assem_irua + (0x27 << 1) + 1, bitgyrc0[id]);
                };
                if  (out & 0x0001) != 0) {
                        poke8(assem_irua + (0x1b << 1) + 1, gyroA_angle_D2),
                        poke8 assem_irua + (0x1c << 1) + 1, gyroA_angle_D1),
                        poke8 assem_irua + (0x1d << 1) + 1, gyroA_angle_D0),
                        poke8 assem_irua + (0x28 << 1) + 1, bitgyrd1[id]);
                        poke8 assem_irua + (0x29 << 1) + 1, bitgyrd0[id]);
                }
        /* mark data according to computed valid bits */
        poke8(assem_irua + (0x10 << 1) + 1, iru_datavalid[0])

    if  (out & 0x000f) != 0) {
        /* indicate data written not just status words *
    };      card[id].assem_write_done = 1;

    };

    if  (id == 1  && ((assem_irub_write_ok == 1) || (afc == 0)))

         *
         ● OK to write to assembly unit, if the flight computer
         * starts a transaction while in this block, we should be
         : able to finish before the assembly unit actually responds
         * to the read but we will protect ourselves by marking the
         , data invalid during the write process
          */

        /* poke8(assem_irub + (0x10 << 1) + 1, iru_datainvalid);  */

        if  (out & 0x0010) != 0) {
                poke8(assem_irub + (0x11 << 1) + 1, iru_gyrosatr[1]);
                poke8(assem_irub + (0x1e << 1) + 1, iru_timetag1[id]);
                poke8(assem_irub + (0x1f << 1) + 1, iru_timetag0[id]);
                poke8(assem_irub + (0x20 << 1) + 1, bit_stat[id]);
                poke8(assem_irub + (0x21 << 1) + 1, bit_crnt[id]);
                poke8(assem_irub + (0x2a << 1) + 1, bit_cpu1[id]);
                poke8(assem_irub + (0x2b << 1) + 1, bit_cpu0[id]);
                poke8(assem_irub + (0x2c << 1) + 1, bit_psm1[id]);
                poke8(assem_irub + (0x2d << 1) + 1, bit_psm0[id]);
                poke8(assem_irub + (0x2e << 1) + 1, bit_dnld[id]);

        if   out & 0x0008) != 0) {
                poke8(assem_irub + (0x12 << 1) + 1, gyroB_angle_A2);
                poke8(assem_irub + (0x13 << 1) + 1, gyroB_angle_A1);
                poke8(assem_irub + (0x14 << 1) + 1, gyroB_angle_A0);
                poke8(assem_irub + (0x22 << 1) + 1, bitgyra1[id]);
                poke8(assem_irub + (0x23 << 1) + 1, bitgyra0[id]);

        if   out & 0x0004) != 0) {
                poke8(assem_irub + (0x15 << 1) + 1, gyroB_angle_B2);
                poke8(assem_irub + (0x16 << 1) + 1, gyroB_angle_B1);
                poke8(assem_irub + (0x17 << 1) + 1, gyroB_angle_B0);
                poke8(assem_irub + (0x24 << 1) + 1, bitgyrb1[id]);
                poke8(assem_irub + (0x25 << 1) + 1, bitgyrb0[id]);
        }.
        if ((out & 0x0002) != 0) {
                poke8(assem_irub + (0x18 << 1) + 1, gyroB_angle_C2);
                poke8(assem_irub + (0x19 << 1) + 1, gyroB_angle_C1)
                poke8(assem_irub + (0x1a << 1) + 1, gyroB_angle_C0)
                poke8(assem_irub + (0x26 << 1) + 1, bitgyrc1[id]);
                poke8(assem_irub + (0x27 << 1) + 1, bitgyrc0[id]);
        }.
        if ( out & 0x0001) != 0) {
                poke8(assem_irub + (0x1b << 1) + 1, gyroB_angle_D2);
                poke8(assem_irub + (0x1c << 1) + 1, gyroB_angle_D1);
                poke8(assem_irub + (0x1d << 1) + 1, gyroB_angle_D0);
                poke8(assem_irub + (0x28 << 1) + 1, bitgyrd1[id]);
                poke8(assem_irub + (0x29 << 1) + 1, bitgyrd0[id]);
         .

        /* mark data according to computed valid bits */
        poke8(assem_irub + (0x10 << 1) + 1, iru_datavalid[1])
```

```
            f ((out & 0x000f) != 0) {
                    /* indicate data written not just status words *
                    card[id].assem_write_done = 1;
             ,
            };
   ,

double
iru_noise(i)
        int             i;

{
        iru_noise_index[i] = ++iru_noise_index[i] & 0x03ff;
        return (iru_noise_array[iru_noise_index[i

}.

int
update_time(i)
        int             i;



        unsigned short int TIMETAG;

        /**
          * Calculate IRU time TIMETAG1 and TIMETAG0
         **/

        iru_timetag[i] = iru_timetag[i] + iru_cycle_time;
        if (iru_timetag[i] > iru_timetag_rollover) {
                /* we have bits to spare so ">" vs ">=" does not matter *,
                iru_timetag[i] -= iru_timetag_rollover;
        };
        /**
        TIMETAG = (iru_timetag[i] / 1000.0) * 1.0e6;
        **/
        TIMETAG = iru_timetag[i] / iru_timetag_sf;
        return (TIMETAG);
};

int
read_iru_power(base)
        char            *base

{

        int             pwr_status;

        pwr_status   peek8(base    0x8000 + 1))

        return (pwr_status


};

int
reset_board(base)
        char            *base,

{

        int             dummy;

        dummy   poke(ba e   (0x8002 + 1)  0xee)           * data is don t care */
        return (0) ;


};

int
ch eck_statu s_register (board)
        int              board;


        card [board] .pwr = read_ iru_power ( *card [board] .base_addr) ;

        cardboard] assem_RTIOU_c t--;

        if ( ( (card [board] .last_pwr & 0x04) == 0x00) && ( (cardboard] .pwr & 0x041 != 0x00
)) {
                /* device just turned on   send reset */
                reset_board( *card [board] .base_addr) ;
        ,,

        if ((card [board] .pwr & 0x04! == 0x00) {
                card [board] assem_insync = 0 ;
                cardboard; assem_RTI OU_ct = cardboard! R TIOU_res et_lim it;
                card [board] consecutive_re cycles = 0;
        .;

        if ( ( (card[board] last_pwr  & 0x08) . . 0) && ((car d[board! .pwr & 0x08) != 0)) !
                / * AFC has started an assembly card transaction */
                card [board] assem RTIOU i nt _pt = cardboard! assem_RTIOU_c t;
                if (cardboard) .assem_write_ok .= 1) !
                        / * AFC transactio n occurred when writes not inhibited * /
                        card [board] .assem_poss ible_parity_ct+ + ;
                        checkpoint 2_card [board] = card [board] ;
                };
        };

        if  ( ( (card [board! las t_pwr & 0x08)  != 0) && ((card[board]. pwr & 0x 08) == 0)) {
                /* AFC has completed an assembly card transaction */
                checkpoint_ card [board] = cardboard! ;
                if ( (cardboard! .assem_RTIOU_ct < card [board] R TIOU_l ockout_value) '
                    (card[board] consecut ive_recycles > cardboard! consecut ive_recycles
_limit)) {
                        f'
                         , if we have recycled, ie assem_RTIOU_ct is  near
                         ' R TIOU_rese t_limit not 0, do not reset - assume we
                         ' are in sync but count consecutive misses after
                         ' limit resync to signal
                         */
                        card[board] .assem_RTIO U_ct  = card [board] .R TI OU reset_limit;
                        cardboard! consecu tive_recycles = 0;
                } else {
                        cardboard] consecut ive_recycles++;
                };
                card [board] assem_insync . 1 ;
                cardboard]  assem_write_done = 0 ;        /* allow another write
                                                           * at proper time● I
                card [board] assem_in t_active++ ;
        };

        card  [board] last_pwr . cardboard; pwr;

        if (card[ board] assem_RTIOU_c t  < card [board] RTIOU_recyc le_value) {
                /"
                 ' either in terrupts are off, or assem card not read, reset
                 * counter and disable interrupt
                 "/
                cardboard! assem_R TIOU_ct = card [board] R TIOU_re se t_lim it;
                card [board] assem_sync loss _ct++;
                card [board] assem_write_done = 0;        /* allow another write
                                                           •at proper  time */
                /* card[ board] .assem_insync = 0; */
                /* poke8 (*card [board! .base_addr + 0x8006 + 1, 0xee) ; '/
        };

        if ( (cardboard) assem_RTIOU_ct > card [board] .RTIOU_interrup t_enable_value)

            (card [board] .assem_RTIOU_ct < card [board] .RTIOU_lockout_value) |
            (card [board] .assem_write_done != 0) ||
            (card [board] assem_insync == 0) ) {
                card[board] .assem_write_ok = 0; /* inhibit writes */
        } else {
                cardboard! .assem_write_ok = 1; /* enable writes '/
        };

        return 0;
};


int
check_ iru_status_reg ister ! )

        iru_pwr ! C ] = read_ iru_power ( assem_irua ) ;
        iru_pwr ! 1] = read_ iru_power (assem_irub) ;

        if (((last iru_pwr[0] & 0x08) == 0) && ((iru_pwr [0] & 0x 08) != 0)) {
                /* AFC has started an assembly card transaction */
                assem_irua_R TIOU_i n pt = assem_irua_RT IOU_ct :
                if (assem_irua_wr ite_ok == 1) {
                        /' AFC tran saction occurred when wr ites not inhibited '/
                        assem_irua_possi ble_parity_ct ++;
                };
        };

        if (((last_iru_pwr [0] & 0x08) != 0) && ((iru_pwr[0] & 0x08) == 0)) {
                /' AFC has completed an assembly card Transaction */
                assem_i rua_RTIOU_ct  =iru_RTIOU_res et_limit;
                assem_i rua_insync . 1 ;
                assem_i rua_int_active++ ;
        };

        if ( ( (las t_iru_pwr[1] & 0x08) == 0)  && ((iru_pwr[1] & 0x08) != 0)) !
                /* AFC has started an assembly card Transaction. */
                assem_i rub_ RTIOU_int_pt = assem_i rub_ RT IOU_ct;
                if (assem_i rub_wri te_ok == 1) {
                        /' AFC transaction occurred when writes not inhibited ' /
                        assem_i  rub>oss  ibLeSarity..ct++;
                };
        };

        if (((last_iru_pwr [1]  & 0x08) != 0) && ((iru_pwr [1] & 0x08) . . 0)) {
                /* AFC has completed an assembly card transaction */
                assem_i rub_R TIOU_ct = iru_RTIOU_reset_l imit;
                assem_irub_insync = 1 ;
                assem_i rub_int_active++;
        );

        las t_iru_pwr [0] = iru_pwr[0] ;
        last _iru_pwr [1] . iru_pwr[1] ;
```

```c
        if ( assem_irua_RTIOU_ct-- < iru_RTIOU_recycle_value ) (
                /*
                 •either interrupts are off, or assem card not read, reset
                 " counter and disable interrupt
                 "1
                assem_irua_RTIOU_ct = iru_RTIOU_reset_limit;
                assem_irua_syncloss_ct++;
                /' assem_irua_insync = 0; */
                /* poke8(assem_irua + 0x8006 + 1, 0xee) ; '/
        };
        if ( assem_irub_RTIOU_ct-- < iru_RTIOU_recycle_value ) !
                /'
                 ' either interrupts are off, or assem card not read, reset
                 •counter and disable interrupt
                 */
                assem_irub_RTIOU_ct = iru_RTIOU_reset_limit;
                assem_irub_syncloss_ct ++;
                / •assem_irub_insync = 0; */
                /* poke8(assem_irub + 0x8006 + 1, 0xee) ; '/
        };

        i< ( ( assem_irua_RTIOU_ct > iru_RTIOU_interrupt_enable_value ) '
            (assem_irua_RTIOU_ct < iru_RTIOU_lockout_value ) '
            (assem_irua_insync == C) ) {
                assem_irua_write_ok = 0;        /' inhibit writes '/
        } else !
                assem_irua_write_ok . 1 ;       /* enable writes */
        };

        if ( ( assem_irub_RTIOU_ct > iru_RTIOU_interrupt_enable_value )
            ( assem_irub_RTIOU_ct < iru_RTIOU_lockout_value )  ''
            (assem_irub_insync == 0 ) ) {
                assem_irub_write_ok . 0 ;       /* inhibit writes */
        ! else {
                assem_irub_write_ok = 1 ;       /* enable writes '/
        };

        return 0 ;
};

int
iru_integrator ( )
{
        int             i, j;

        for (i = 0; i < 6; i++){
                check_status_register ! i ) ;
        };

        iru_1024_hz_cycle++ ;
        iru_1024_hz_cycle . iru_1024_hz_cycle & 0x000003ff;

        if(iru_cycle_array [iru_1024_hz_cycle] ' = 0) {
                iru_channel = (iru_channel + 1) & 0x00000003;
                iru_sim ( iru_channel ) ;
        } else {
                if (iru_channel == 3) { /** safe to pick up dynamics rates **/
                        for (i = 0; i < 2; i++) !
                                if (dyn_rate_avail [i] '= 0) (
                                        for (j = 0; j < 4; j++) {
                                                angular_rate[i][j] = dyn_iru_rate[i][j] ;
                                        };
                                        dyn_rate_avail [i] = 0;
                                };
                        };
                };
        };
        return 0;
};

int
iru_sum_angle ( id, i )
        int             id;
        int             i;



        double          angle_rate;
        unsigned short int time;

        if (i ==0){             /** time and sat. bits are synced to start of update
 cycle **/
                iru_gyrosatr [id] = 0x00;       I*' clear saturation bits **/
                time = update_time(id) ;
                iru_timetag0[id] = time & 0x00ff;
                iru_timetag1[id] = time >> 8;

        angle_rate = angular_rate[id][i] ;

        /**
         ** if angular rate exceeds 15 deg/sec set coorsponding
         •+ gyro saturation bit
         ...

        if(fabs(angle_rate)> (scaled_iru_gyro_sat_limit) ) !
                if (id . . 0){
                        Jim_gyr_stra ++;
                } else {
                        Jim_gyr_strb++ ;

                iru_gyrosatr[id] = iru_gyrosatr[id]' gyro_sat_mask[i] ;
                DEangle[id][i] = 0.0;   /** set integrated angle to 0 **/
                iru_gyrosatr_delay[id][i] . iru_gyrosatr_delay_time;
                trap_angular_rate_a = angular_rate[ id! [0] :
                trap_angular_rate_b = angular_rate [ id][1] ;
                trap_angular_rate_c = angular_rate ! id! [2];
                trap_angular_rate_d = angular_rate[id][3] ;
                trap_i . i ;
                trap_mask = iru_gyrosatr[id] ;
                trap_angle_rate = angle_rate;
        } else {
                if ( (--iru_gyrosatr_delay[id][i] ) <= 0) {
                        iru_gyrosatr_delay[id][i] = 0;
                        /** apply cycle delta time and scale factor **/
                        DEangle[id][i] += (angle_rate•iru_cycle_time_sf ) ;
                        if (DEangle[ id! [i] >= iru_rollover) {
                                DEangle[id][i] -= iru_rollover;
                        } else {
                                if (DEangle[id][i] <= -iru_rollover) {
                                        DEangle[id][i] += iru_rollover;


                } else {        /** keep sat flag condition until timer expires **/
                        iru_gyrosatr[id] . iru_gyrosatr[id] 'gyro_sat_mask[i];
                        DEangle[id][i] = 0.0;   /** set integrated angle to 0 **/
                ,,
        };
        if ( (iru_noise_on '= 0) && ( (iru_gyrosatr[id] & gyro_sat_mask[i] ) == 0) )
                Eangle[id][i] = (DEangle[id][i] + iru_noise(id) ) / iru_angle_sf;
        else
        Eangle[id][i] = DEangle[id][i] / iru_angle_sf;

    iru_datavalid[id] = 0x00bc ^ iru_gyrosatr[id];  /** bit 7 always = 1 **/

    if(id == 0) {

        switch (i) {

        case 0:
                gyroA_angle_A0 = Eangle[id][0] & 0x000000ff;
                gyroA_angle_A1 = (Eangle[id][0] >> 8) & 0x000000ff;
                gyroA_angle_A2 = (Eangle[id][0] >> 16) & 0x 000000ff;
                break;

        case 1:
                gyroA_angle_B0 = Eangle[id][1] & 0x000000ff;
                gyroA_angle_B1 = (Eangle [id][1] >> 8) & 0x000000ff;
                gyroA_angle_B2 = (Eang le [id][1] >> 16) & 0x000000ff;
                break:

        case 2:
                gyroA_angle_C0 = Eangle[id][2] & 0x 000000ff;
                gyroA_angl e_C1 = (Eang le [id][2] >> 8) & 0x 000000ff;
                gyroA_angle_C2 = (Eangle [id][2]  >> 16) & 0x000000ff;
                break.;

        case 3:
                gyroA_angle_D0 = Eangle[id][3] & 0x000000ff;
                gyroA_angle_D1 = (Eangle[id][3] >> 8) & 0x000000ff;
                gyroA_angle_D2 = (Eangle[id][3] >> 16) & 0x000000ff;
                break;
        },

    };

    if (id == 1) {


        switch (i) {

        case 0:
                gyroB_angle_A0 = Eangle[id][0] & 0x000000ff;
                gyroB_angle_A1 = (Eangle[id][0] >> 8) & 0x000000ff;
                gyroB_angle_A2 = (Eangle[id][0] >> 16) & 0x000000ff;
                break;

        case 1:
                gyroB_angle_B0 = Eangle[id][1] & 0x0000ff;
                gyroB_angle_B1 = (Eangle[id][1] >> 8) & 0x000000ff;
                gyroB_angle_B2 = (Eangle[id][1] >> 16) & 0x000000ff;
                break;

        case 2:
                gyroB_angle_C0 = Eangle[id][2] & 0x000000ff;
                gyroB_angle_C1 = (Eangle[id][2] >> 8) & 0x000000ff;
                gyroB_angle_C2 = (Eangle[id][2] >> 16) & 0x000000ff;
                break;

        case 3 :
                gyroB_angle_D0 = Eangle [id][3] & 0x 000000ff;
                gyroB_angle_D1 = (Eangle[id][3] >> 8) & 0x0000 00ff;
                gyroB_angle_D2 = (Eangle[ id][3] ">> 16) & 0x000000ff;
                break;
        };
        };

};


int
iru_sim ( target_channel )
    int             targe t_channel; /" 0=chan A, 1=chan B, 2.than C, 3.than D *
*/
{

    int             i;
    int             Sft Rst[2] ;      /** Soft Reset Function code ** /
    /**int  angular_ rate [2][4] ; ●*/
    /**  i ru_mode:  0 = power off to on  transition,
                     1 = power turned on - doing BIT 1 sec timer,
                     2 = wait for download to complete,
                     3 = soft reset,
                     4 = normal **/


    SftRst[0] = peek8(assem_irua + (0x80 << 1) + 1);

    if ((SftRst[0] != 0x03) ||
        (peek8(assem_irua + (0x81 << 1) + 1) != 0x00) ||
        (peek8(assem_irua  + (0x82  <<  1) + 1) != 0x00)
        (peek8(assem_irua  + (0x83  <<  1)  + 1) != 0x00)
        (peek8( assem_irua  + (0x84  <<  1) + 1) != 0x00)
        (peek 8(assem_irua  + (0x85  <<  1) + 1) != 0x00)
        (peek8(assem_irua + (0x86 << 1) . 1) != 0x00) ) /
            /* die? not pass reset packet test */

            SftR st[0] = 0;
    };

    Sft Rst[1] = peek8(assem_irub + (0x80 << 1) + 1) ;

    if ((SftR st[1]  != 0x03) ||
        (peek8(assem_irub + (0x81  <<  1) + 1) != 0x 00)
        (peek8(assem_irub + (0x82 << 1)  +  1) != 0x00)
        (peek8(assem_irub  .  (0x83 << 1)  .  1) != 0x00)
        (peek  8(assem_irub + (0x84  <<  1) + 1) != 0x00)
        (peek8(assem_irub  + (0x85  <<  1) + 1) != 0x 00)
        (peek8(assem_irub + (0x86 << 1) + 1) != 0x00) 1 /
            /* did not pass reset packet test● I
            SftRst[1] = 0;
    );


    for (i = 0; i < 2; i++) {

            if ((card[i].pwr & 0x04) == 0) {        /** device is off **/
                    iru_reset(i);
                    bit_crnt[i] = 0x00;
                    bit_dnld[i] = 0x00;
                    dwn_ld_done[i] = 0x00;
                    iru_mode[i] = 0;
            } else {        /** device is on **/

                    switch (iru_mode[i]) {

                    case 0:/** device just turned on **/
                            iru_reset(i);
                            iru_rtiou_write(i, 31, force_write);    /** initiali
ze iru data **/
```

```
                        iru_timer[ i] . iru_BIT_t ime;     /** counts down to zero
**/

                        iru_mode [i] = 1;
                        break;

                case 1: /** 1 second BIT timer **/
                        if (iru_timer [i] > 0) {
                                iru_timer[i] --;
                        } else {          /** timer expired **/

                                bit_dnld[i] . 0x30;      /' BIT complete● I
                                iru_gyrosa tr [i] = 0x00;
                                bit_crnt[i] = 500 / 15.625;
                                dwn_ld_done [ i ] = 0x05;
                                iru_rt iou_write (i, 16, force_write) ;     / ** upda
te status only **/

                                iru_mode [i] = 2 ;
                        };
                        break;

                case 2:/** wait for download to complete **/

                        /** check for soft reset  ...

                        if (S ftRst[i] == 0x03) !         /** soft reset just occu
rred **/

                                iru_rese t(i) ;
                                iru_rti ou writ e(i, 31, force write)     / ** upda
te all **/

                                iru_timer[i] = iru_soft_reset_time;     /** coun
ts  down  to zero**/

                                iru_m ode [i ! = 3 ;
                        ' else !

                                if (dwn_ld_done[i] == 0x0f) {   /** download com
plete **/

                                        iru_mode[i] . 4;
                                ! else {        /' •check for download complete
...

                                        iru_download( i ) ;
                                1;
                        };
                        break;

                case 3:/** soft reset delay 1  second  timer ● +1

                        if (iru_timer[i] > 0) !
                                iru_timer [i]--;
                        ! else {          /** timer expired **/

                                /**
                                 ** this prevents a continous so ftreset
                                 ** by writing over the function code
                                ●*,

                                if (i == 0) {
                                        poke8(assem_irua + (0x80 << 1) + 1, 0xff
);

                                };

                                if (i  ==1) {
                                        poke8 (assem_irub + (0x80  << 1) + 1, 0xff
);

                                };
```

```
                                        iru_rt iou_write (i, 15, force_write);     /**
update status only **/

                                        soft_reset_flag[i] = 1;

                                        if (dwn_ld_done [i] == 0x0f) {    /** download
 complete **/

                                                iru_mode [i] = 4;
                                        ? else !          /** check  for download compl
ete● +1

                                                iru_mode [i] = 2;
                                        };
                                };
                                break;

                        case 4: /** normal opera tion **/

                                /**   check   for soft reset first before calculating an
gles **/

                                if (SftRst[i] == 0x03) {          /** soft reset just
occurred **/

                                        iru_reset(i) ;
                                        iru_rtiou_write( i, 31, force_write) ;     /**
update all **/

                                        iru_timer [ i]  = iru_soft_rese t_t ime;     /**
counts down to zero ** I

                                        iru_mode[i] = 3;
                                } else {
                                        i ru_sum_angle ( i ,  target_channel ! ;
                                        if ( target_channel == 3 ) !
                                                /** write to iru after all angles co
mplete'* /

                                                iru_rtiou_write(i, 31, check_afc);
                                        };
                                };

                                break;
                        };
                };
                if (iru_mode[i] != 4)  !
                        card [i].assem_insync = 0 ;
                };



};

int
iru_download(id)
        int              id;



        if ((id == 0) && (fake_download != 1)) {

                FuncCodeA . peek R (assem_irua + (0x80 << 1) + 1);

                if (FuncCodeA == 0x09) !
                        MspaceA = peek8 (assem_irua + (0x80 << 1) + 5) ;
                        if (MspaceA == 0x00) !
                                /* 0x00 space GSP p-space ' /
                                bit_dnld [id] = bit_dnld[id] ! 0xc0;
                                poke8(assem_irua + (0x2e << 1) + 1, bit_dnld[id] ! ;
                                /* chksum on p-space good * /
                        dwn_ld_done [ id! = dwn_ld_done [ id!  | 0x03 ;
                };

                if (MspaceA  ==  0x10) {
                        /* 0X00 space d-space * /
                        bit_dnld[id] . bit_dnld[id] ' 0x0c;
                        poke8 (assem_irua + (0x2e << 1) + 1, bit_dnld [id] ) ;
                        /* chksum on d-space good * /
                        dwn_ld_done [ id! . dwn_ld_done [ id] | 0x0c;
                };

        };

    };


    if ((id == 1) && (fake_ download != 1)) {

        Func Code B = peek 8 (assem_irub + (0x80 << 1) . 1) ;

        if (Fix.cC0de2 == 0x09  )  {
                Mspace B = peek8 (assem_irub . (0x80 << 1 ) + 5 ) ;
                if (MspaceB == 0x00) !
                        /* 0x00  space  GSP p-space*/
                        bit_dnld [id] . bit_dnld[id] | 0xc0;
                        poke8 (assem_irub + (0x2e << 1) + 1, bit_dnld [id] ) ;
                        / •chksum  on  p-space  good•/
                        dwn_ld_done [ id] = dwn_ld_done [ id! 0x03  ;
                };

                if  (MspaceB == 0x10)  {
                        /* 0x00 space d-space */
                        bit_dnld[id] = bit_dnld[id] ' 0x0c;
                        poke8(assem_irub + (0x2e << 1) + 1, bit_dnld[id]);
                        /* chksum on d-space good */
                        dwn_ld_done[id] = dwn_ld_done[id] | 0x0c;
                };

        },


    if( fake_download  ==  1) {
        bit_dnld[ id] = 0xfc;
        iru_da tavalid[ id] . 0xbc;
        dwn_ld_done [id] . 0x0f ;
    };

}

int
run_assem()
{
    rwa_loop ( ) ;
    return 0;
}
```

```
/* ega_model.c */
/* cassini itl ega model *

/* for the d_real typedef */
#include "types-darts.h"

/* for matrix subroutines *
#include "linalg-darts.h"

/* standard */
#include <string.h>

/* local */
#include "ega_model.h"

/* Diagnostics */
extern int      model;
extern int      diag_time;
extern int      diag_level
extern FILE    *f_ega;
extern int      diag_flag;
extern d_real   bb_timetag

/* local functions */
void            ega_kinematics(), ega_extension(), l2solve()

/* namelist data */
d_real          ega_max_delta, ndlvdt, ega_null_length, ega_err_tol, p_joint[2][2][3], u
_joint[2][2 [3],
                me2_null_trans[3][3], me1_null_trans[3][3];
int             ega_counter_max;

/* local to model */
d_real          ega_curr_ext[4];
int             ega_onoff_a, ega_onoff_b;

/** Modification 29 July 96 J. Bunn:
Changed onoff_a(b) to ega_onoff_a(b) to avoid model conflicts
Changed default to ON
**/

void
ega_init
{
        int             i

        /* namelist parameters *

        ega_null_length = 9.8;
        ega_max_delta = 0.05667;
        ega_max_delta = 0.01;
        ndlvdt = 4.;

        /*
         * mei_null_trans are the transformations from engine to spacecraft
         * coords
         */

        me1_null_trans[0][0] = 0.99863117108034;
        me1_null_trans[0][1] = 0.00000000000000;
        me1_null_trans[0][2] = -0.05230472394256
        me1_null_trans[1][0] = 0.00496283907661;
        me1_null_trans[1][1] = 0.99548841288797;
        me1_null_trans[1][2] = 0.09475331146768;
        me1_null_trans[2][0] = 0.05206874662412;
        me1_null_trans[2][1] = -0.09488319032258;
        me1_null_trans[2][2] = 0.99412575955923;

        me2_null_trans[0][0] = 0.99863792821450;
        me2_null_trans[0][1] = 0.00000000000000;
        me2_null_trans[0][2] = -0.05217555300558;
        me2_null_trans[1][0] = -0.00615247411855;
        me2_null_trans[1][1] = 0.99302325091709;
        me2_null_trans[1][2] = -0.11775810035948;
        me2_null_trans[2][0] = 0.05181153726400;
        me2_null_trans[2][1] = 0.11791871411295;
        me2_null_trans[2][2] = 0.99167068196467;

        ega_err_tol = 1.e-7;
        ega_counter_max = 6;

        /* REA-1 */
        p_joint[0][0][0] = 0.09826216260427;
        p_joint[0][0][1] = 0.19326857152374;
        p_joint[0][0][2] = -0.24259124338664;
        p_joint[0][1][0] = -0.09700628079466;
        p_joint[0][1][1] = 0.19342412374100;
        p_joint[0][1][2] = -0.24261331226352;
        u_joint[0][0][0] = 0.77279074868430;
        u_joint[0][0][1] = -0.48475699251892;
        u_joint[0][0][2] = -0.53457785616215;
        u_joint[0][1][0] = -0.77526482865983;
        u_joint[0][1][1] = -0.48079894849746;
        u_joint[0][1][2] = -0.53492556383186;

        /* REA-2 */
        p_joint[1][0][0] = 0.09952277331660;
        p_joint[1][0][1] = -0.19265782170509;
        p_joint[1][0][2] = -0.24251971359319;
        p_joint[1][1][0] = -0.09493540652512;
        p_joint[1][1][1] = -0.19306247518653;
        p_joint[1][1][2] = -0.24270448158418;
        u_joint[1][0][0] = 0.76977251224720;
        u_joint[1][0][1] = 0.48934508284425;
        u_joint[1][0][2] = -0.53489852290059;
        u_joint[1][1][0] = -0.77823654184649;
        u_joint[1][1][1] = 0.47609528989577;
        u_joint[1][1][2] = -0.53455712336594;

        /* initialize */
        for (i = 0; i < 4; i++)
                ega_curr_ext[i] = 0.;

        /* default on */
        ega_onoff_a = 1;
        ega_onoff_b = 1;
}

void
ega_onoff(onoff_str)
        char            onoff_str[]
{
        if (!strcmp(onoff_str, "egaa on"))
                ega_onoff_a = 1;
        if (!strcmp(onoff_str, "egaa off")
                ega_onoff_a = 0;
        if (!strcmp(onoff_str, "egab on"))
                ega_onoff_b = 1;
        if (!strcmp(onoff_str, "egab off")
                ega_onoff_b = 0;
```

```
)

/*
 · Indexing: [(+Y REA/+X EGA) , ( (+Y REA/-X EGA) , (-Y ?. EA/+X EGA) , (-Y P. EA/-X
 · EGA ) ]
 •
 · Note that within the CAB documentation, the +Y REA is designate? as the A REA
 • and the -Y REA is designated as the B REA.  Additionally, the +X EGA is
 • designated as EGAPA, while the -X EGA is designated as EGAQA.
 ·
 · mel_u_vec:    Unit vector of +Y ME thrust in S/C coords me2_u_vec :     Un i t
 • vector of -Y ME thrust in S/C coords
 */

void
ega_model ( ega_ext_c om, lvdt_pos_est, me1_u_vec, me2_u_vec )
        d_rea l           ega_ext_com [ ] , lvdt_pos_est [ ] ,  me1_u_vec [ ] , me2_u_vec [ ! ;

/*
 * in:  ega_ext_c om   commanded extensions lvdt_pos_est lvdt  extensions
 *
 * out  me 1_u_v e c     main engine vectors me2_u_v ec
 */

{
        d_rea 1           delta;
        int               i;

        if (ega_onoff_a) {


                /* update state per command -- limit if necessary: */
                for (i = 0; i < 2; i++) {
                        ega_ext_com[i * 2 + 0] = ega_ext_com[i * 2 + 0] * .00027246;
                        lvdt_pos_est[i * 2 + 0] = lvdt_pos_est[i * 2 + 0] * .00027246;
                        delta = ega_ext_com[i * 2 + 0] - ega_curr_ext[i * 2 + 0];
                        if (delta > ega_max_delta)
                                ega_curr_ext[i * 2 + 0] += ega_max_delta;
                        else if (delta < -ega_max_delta)
                                ega_curr_ext[i * 2 + 0] -= ega_max_delta;
                        else
                                ega_curr_ext[i * 2 + 0] = ega_ext_com[i * 2 + 0];


                I * check lvdt data * /
                for (i = 0; i < 2; i++) {
                        if ( fabs ! lvdt_pos_est [i ' 2 + 0] - ega_curr_ext [i  * 2 + 0]) > nd
lvdt * ega_max_delta )
                                ega_curr_ext[i * 2 + 0] = lvdt_pos_est[i * 2 + 0];



        }
        if (ega_onoff_b) {

                /* update state per command -- limit if necessary: '/
                for (i = 0; i < 2; i+. ) {
                        ega_ext_c om [ i  * 2 + 1] = ega_ext_com[i ' 2 + 1] ' .00027246;
                        lvdt_pos_est[i * 2 + 1] = lvdt_pos_est[i ' 2 + 1]•.  00027245;
                        delta = ega_ext_com[i* 2 + 1] - ega_curr_ext [i * 2 + 1 ] ;
                        if (delta > ega_max_delta)
                                ega_curr_ext [ i ' 2 + 1] += ega_max_del ta;
                        else if (delta < -ega_max_delta)
                                ega_curr_ext [ i " 2 + 1 ] -. ega_max_delta;
                        else
```

```
                                ega_curr_ext ! i •2 + 1] . ega_ext_c om [i  * 2 + 1];
                }

                /' check lvdt data */
                for (i = 0; i < 2; i++) {
                        if ( fabs ( lvdt_pos_est [i , 2 + 1] - ega_curr_ext [i * 2 + 1])
> ndlvdt•ega_max_delta )
                                ega_curr_ext[i ·2 + 1] = lvdt_pos_est [ i * 2 + 1];
                }

        }
        /* kinematics */
        ega_kinematics (me1_u_vec, me2_u_vec ) ;

        /* Diagnostics */
        if (model == 3 && diag_time- - > 0) {
                if (diag_flag == 0) (
                        if  (  (f_ega = fopen(" /si m/tmp/ega_diags ", "w") ) ==(FILE•) N
ULL) !
                                printf ( "File Open Load Faiure for EGA diagnos tics\n"
);
                        };
                        diag_flag = 1;
                }
                if (diag_level == 1 '' diag_level == 2 '' diag_level == 3) {
                        if (f_ega == (FILE *) NULL) {
                                printf("OUTPUTS: me1_u_vec %12.6f %12.6f %12.6f\n
       me2_u_vec %12.6f %12.6f  %12.6f\n", me1_u_vec[0], me1_u_vec[1], me1_u_vec[2],
me2_u_vec[0], me2_u_vec[1], me2_u_vec[2]);
                        } else {
                                fprintf(f_ega, "OUTPUTS: me1_u_vec %12.6f %12.6f %12
.6f\n          me2_u_vec %12.6f %12.6f  %12.6f\n", me1_u_vec[0], me1_u_vec[1], me1_u_
vec[2], me2_u_vec[0], me2_u_vec[1], me2_u_vec[2]);
                        };
                };
                if(diag_level . . 2  '  diag_level == 3) {
                        if (f_ega == (FILE *) NULL) {
                                printf("INPUTS: Est. Com: %12.6f %12.6f %12.6f %12.6
f \n        LVDT:  %12.6f %12.6f %12.6f %12.6f\n", ega_ext_com[0], ega_ext_com[1], e
ga_ext_com[2], ega_ext_com[3], lvdt_pos_est[0], lvdt_pos_est[1], lvdt_pos_est[2], lv
dt_pos_est[3]);
                        } else {
                                fprintf(f_ega, "INPUTS: Est. Com: %12.6f %12.6f %12.
6f %12.6f \n        LVDT:  %12.6f %12.6f %12.6f %12.6f\n", ega_ext_com[0], ega_ext_c
om[1], ega_ext_com[2], ega_ext_com[3], lvdt_pos_est[0], lvdt_pos_est[1], lvdt_pos_es
t[2], lvdt_pos_est[3]);
                        };
                };
                if (diag_level == 3) !
                        if (f_ega == (FILE● ) NULL) {
                                printf( "\n Internals: curr ext: %12.6f %12.6f %12.6f
%12.6f\n            onoff_a %3d  onoff b: %3d\n", ega_curr_ext [0] , ega_curr_ext [ 1 ! , e
ga_curr_ext [2 ] ,  ega_curr_ext [ 3 ! , ega_on of f_a, ega_onof f_b);
                        ! else {
                                fprintf ( f_ega,  "\ nInternals: curr ext: %12.6f %12.6f
 %12.6f %12.6f\n            onoff_a %3d  onoff b: %3d\n" , ega_curr_ext [0 ] , ega_curr_e
xt [1] , ega_curr_ext [2 1 , ega_curr_ext [3 ] , ega_onoff_a, ega_onoff_b);
                        };
                };
                if (f_ega == (FILE● ) NULL) {
                        print f("End of Cyc le.  bb_timetag = %f\n\n" , bb_timetag) ;
                ) else {
                        fprin tf(f_ega," End of Cycle.   bb_timetag = %f\n\n" , bb_time
tag) ;
                };
        };
        if (diag_time . . 0 && model == 3) {
                diag_flag . 1;
                if ( f_ega != (FILE * ) NULL) {
                        if (fclose(f_ega) != 0) !
                                printf ( "Error closing Engine Gimbal Debug File \n" ) ;
                        };
                };
        };
        if (diag_time < 0 && model != 3)
                diag_time = 0;
1

void
ega_kinematics(me1_u_vec, me2_u_vec )
        d_real          me1_u_vec[] ,  me2_u_vec[] ;
{
        /**
         * local:
         *
         * extc (2)                     "calculated" normalized extensions
         * ext_err ( 2 )                normalized extension error
         , angle_ step(2 )              newton step
         *big_g(2 ,2)                   matrices of partials
         ' angl e(2)                    gimbal angles
         ' rate (2)                     output gimbal rates (rad/s)
         , accel (2)                    output gimbal accels (rad/ s/s)
         * ext (2)                      n orma lized extensions
         * ext_dot (2 )                 normalized extension rates
         *ext_dot_dot (2 )              normalized extension accels
         . mtemp( 2,3)                  temporary matrix
         , linmap(2 ,2) and vtemp(2)    temporary vectors
         * i,j,k                        misc counters
         * convergence                  convergence flag
         ' counter                      counter for newton iterations
         ' c1, c2, s1, C2               trig fun ct ions
         • td1d2                        trans matrix
         * geom                         thrust unit vector in engine coords
         , me_str                       temp trans formati on matrix
         */

        d_real          extc[2], ext_err[2] ,  ang le_step[2], angle '2 ] , ext[2] , ext_err_no
rm,
                        c1, c2, s1, 52, geom [3];

        d_real          big_g[2][2] , mtemp[2][3], linmap[2][2], td1d2[3][3], me_str[3][3
];

        int             rea, i, j, convergence,  counter;

        / ' REA loop - thrust vector for each REA*/
        for (rea = 0;  rea < 2; rea++) {

                I * decompose  extension data and normalize*/
                for (i = 0; i  <  2; i++)  r
#if 0
                        ext[i]  . ega_curr_ex t [rea ' 2 + i ] / ega_null_length;
#endif
                        ext [i]  =  ega_curr_ext [ i  * 2 . rea! /ega_null_length;


                /* initialize guess from inverting linear map */
                for (i = 0; i < 2; i++) {
                        cross(mtemp[i], u_joint[rea][i], p_joint[rea][i]);
```

```
                for (i = 0; i < 2; i++) {
                        for (j = 0; j < 2; j++) {
                                linmap[i][j] = mtemp[i][j];
                        }
                }
                l2solve(linmap, ext, angle) ;

                /* newton loop '1
                convergence = 0;
                counter = 0;
                while (convergence == 0 ) {

                        if (counter  > ega_counter_max )  {
                                /**
                                        printf ( "aaaaack!  ega model : kinematics bar fed" !

                                **/
                                convergence = 1;
                        }
                        /* calculate  extension and matrix of partials */
                        ega_extens ion (angle, extc, big_g, rea) ;

                        I" evaluate  extension error */
                        for (i = 0; i < 2; i++) !
                                ext_err[i] = ext[i] - extc [i];
                        }

                        ext_err_norm  .  sqrt (p ow(ex t_err [0] , 2  ) +
                                                pow(ext_err[1], 2.)):

                        /*test convergence * /
                        if (ext_err_norm  .  ega_err_tol! !
                                convergence = 1;
                        }
                        if (convergence ==0)  !
                                /*  solve for newton step  */
                                l2solve(big_g, ext_err, angle_ step) ;

                                /* take newton step */
                                for (i = 0; i < 2; i++) {
                                        angle[i] = angle[i] + angle_step[i];


                                counter = counter . 1 ;

                        }
                }               /* end of newton loop */

                if (model  ==  3 && diag_time > 0 && diag_level  ==  3  && diag_flag == 0
) {

                        if ((f_ega = fopen("/sim/tmp/ega_diags", "w")) == (FILE *) N
ULL) {

                                printf("File Open Failure for EGA diagnostics\n");
                        };
                        diag_flag = 1;
                        if (f_ega == (FILE *) NULL) {
                                printf("REA: %3d Angle0: %12.6f  Angle1:%12.6f\n", r
ea, angle[0], angle[1]);
                        } else {
                                fprintf(f_ega, "REA: %3d Angle0: %12.6f  Angle1:%12.
6f\n", rea, angle[0], angle[1]);
                        };
                };
```

```
            /*compute ME thrust unit vector given solution for angles */
            /* first compute the null_to_rotated transform   */
            s1 = sin(angle[0]);
            c1 = cos(angle[0]);
            s2 = sin(angle[1]);
            c2 = cos(angle[1]);
            td1d2[0][0] = c2;
            td1d2[0][1] = s1 * s2;
            td1d2[0][2] = -c1 * s2;
            td1d2[1][0] = 0.0;
            td1d2[1][1] = c1;
            td1d2[1][2] = s1;
            td1d2[2][0] = s2;
            td1d2[2][1] = -s1 * c2;
            td1d2[2][2] = c1 * c2;

            geom[0] = 0.0;
            geom[1] = 0.0;
            geom [2] = -1.0;

            /* compute u3 vector in spacecraft coordinates */
            if (rea == 0) {
                    times_33mat33mattran(me_str, me1_null_trans, td1d2
                    times_33mat3vec(me1_u_vec, me_str, geom);
            } else {
                    times_33mat33mattran(me_str, me2_null_trans, td1d2
                    times_33mat3vec(me2_u_vec, me_str, geom);
            }
    }                       /* end of rea loop */


void
ega_extension(angle, ext, big_g, rea)
        d_real          *angle, *ext;
        d_real          big_g[2][2];
        int             rea;

/*
 * In:     angle[2] :        engine gimbal angles (rad)
 *
 *
 * Out:    ext[2]:          normalized ega extensions big_g[2][2]:    partial
 * derivative matrix rea:            number of the engine assembly under
 * consideration
 */

{


        /**
         " local:
         •
         ' s1                    sin  of angle(1) c1                        cos
         ' of angle(l) s2                        sin  of angle(2) c2 os  of
         * angle(2)
         '
         •alpha(2)               intermedi ate    9(2,2)                    alpha
         ' derivatives h(2, 2,2)         second alpha  derivatives
         '
         • stemp1->2             temporary scal ars vtemp1 (3) ary vector
         •mtemp1->4 (3,3)                temporary matrices pmj (2,3) _join t -
         ' u_j oint gamma (3, 3)         identity minus rotation matrix
         * gammat (3,3)          gamma transpose dgamma (2 , 3,3 ) erivative of
         * gamma dgammat (2,3,3)         dgamma transpose i, j , k
         " isc counters
         •I
```

```
        d_real          s1, c1, s2, c2, alpha[2], stemp1, stemp2, vtemp1[3];

        int             i, j;

        d_real          gamma_local[3][3], gammat[3][3], g[2][2], mtemp1[3][3], mtem
p2[3][3],
                        mtemp3[3][3], pmj[2][3], dgamma[2][3][3], dgammat[2][3][3];

        s1 = sin(angle[0]);      /* trig functions */
        c1 = cos(angle[0]);
        s2 = sin(angle[1]);
        c2 = cos(angle[1]);

        gamma_local[0][0] = 1. - c2;     /* form gamma and its derivatives */
        gamma_l ocal[0][1] = -s1 ' s2;
        gamma_l ocal [0] [2] = c1 * 52;
        gamma_l ocal[1][0] = 0.;
        gamma_l ocal[1] [1] = 1. - c1;
        gamma_l ocal[1][2] = -s1;
        gamma_local[2][0] = -s2:
        gamma_local[2] [1] = s1 * c2;
        gamma_local[2][2]  .  1.  -  c1•c2;

        dgamma[0][0][0]  = 0.;
        dgamma[0][0][1]  = -c1 * s2;
        dgamma[0][0][2]  = -s1 * s2;
        dgamma[0][1][0]  = 0.;
        dgamma[0][1][1]  = s1;
        dgamma[0][1][2]  = -c1;
        dgamma[0][2][0]  = 0.;
        dgamma[0][2][1]  = c1 * c2;
        dgamma[0][2][2]  = s1 * c2;

        dgamma[1][0][0]  = s2;
        dgamma[1][0][1]  = -s1 * c2;
        dgamma[1][0][2]  = c1 * c2;
        dgamma[1][1][0]  = 0.;
        dgamma[1][1][1]  = 0.;
        dgamma[1][1][2]  = 0.;
        dgamma[1][2][0]  = -c2;
        dgamma[1][2][1]  = -s1 * s2;
        dgamma[1][2][2]  = c1 * s2;


        /* calculate the alpha's */
        transpose_33mat(gammat, gamma_local);
        for (i = 0; i < 2; i++) {
                times_33mat3vec(vtemp1, gammat, p_joint[rea [i]);
                minus_3vec(pmj[i], p_joint[rea][i], u_joint rea][i]);
                stemp1 = dot3(vtemp1, pmj[i]);
                stemp2 = dot3(vtemp1, vtemp1);
                alpha[i] = -stemp1 + 0.5 * stemp2;


        /* extensions */
        for (i = 0; i < 2; i++) !
                ext[i]  = sqrt (1. + 2. , alpha[i]) - 1.;


        /* calculate derivative matrices */

        /* get g  */
        for (j = 0; j < 2; j++) {
                transpose_33mat(dgammat[j] , dgamma ! j] ) ,
        }
        for (i = 0; i < 2; i++) {
                for (j = 0; j < 2; j++) {
                        times_33mat3vec(vtemp1, dgammat[j], p_joint[real[i]);
                        stemp1 = dot3(vtemp1, pmj[i]
                        times_33mat33mat(mtemp1, dgamma[j], gammat ,
                        transpose_33mat(mtemp2, mtemp1);
                        plus_sqmat(&mtemp3[0][0], &mtemp1[0][0],
                                   &mtemp2[0][0]  3),
                        times_33mat3vec(vtemp1, mtemp3, p_joint[real[i] ,
                        stemp2 = dot3(vtemp1, p_joint[real[i]);
                        g[i][j] = -stemp1 + 0.5 * stemp2;
                }
        }

        /* get big_g */
        for (i = 0; i < 2; i++) {
                for (j = 0; j < 2; j++) {
                        big_g[i][j] = g[i][j]    1  + ext[i]
                }
        }

}                               * end of ega_extension *

void
l2solve(a, b, angle)
/*
 * solves 2x2 ax = b input a[  [2] b[2] output angle[2]
 */
        d_real          *b, *angle;
        d_real          a[2][2];


        d_real          det;

        det = a[0][0]   a[1][1]   a[1][0] * a[0][1],

        if (det == 0.) {
                printf("aaaaack! ega model: 2x2 map singular");

        angle[0] = (a[1][1] * b[0] - a[0][1] * b[1]) / det;
        angle[1] = (-a[1][0] * b[0] + a[0][0] * b[1]) / det;
}
```

gyro_model.c

```
/* gyro_model.c */
/* cassini itl gyro model *

/* for the d_real typedef */
#include "types-darts.h"

/* for matrix subroutines */
#include "linalg-darts.h"

/* standard */
#include <math.h>
#include <string.h>

/* Diagnostics */
extern int      model;
extern int      diag_time;
extern int      diag_level
extern FILE    *f_gyro;
extern int      diag_flag;
extern d_real   bb_timetag

/* local */
#include "gyro_model.h"

/* namelist data */
d_real          gyro_sens_a1[3], gyro_sens_a2[3]  gyro_sens_a3[3], gyro_sens_a4[3],
                gyro_sens_b1[3], gyro_sens_b2[3]  gyro_sens_b3[3], gyro_sens_b4[3],
                gyro_rate_quant;

/* local */
int             gyro_onoff_a, gyro_onoff_b;
d_real          quant_remainder[8];

/** Modification 29 July 96 J. Bunn:
Changed onoff_a(b) to gyro_onoff_a(b) to avoid model conflicts
**/

void
gyro_init()

        int             i

        /* namelist parameters */

        gyro_rate_quant = 0.5e-6;

        /*
         * Define gyro sense axes.  Ref: IOM 3413-94-010, IRU Location and
         * Orientation: Gyro Input Axis Orientation Doug Bernard July 28 1994
         */

        gyro_sens_a1[0  = 0.418432;
        gyro_sens_a1[1  = -0.908248,
        gyro_sens_a1[2  = 0.;

        gyro_sens_a2[0] = -0.642229;
        gyro_sens_a2[1] = -0.295876;
        gyro_sens_a2[2] = -0.707107;

        gyro_sens_a3[0] = -0.642229;
        gyro_sens_a3[1] = -0.295876;
        gyro_sens_a3[2] = 0.707107;

        gyro sens_a [0] = 0 5;
        gyro sens_a [1] = 0 866025;
        gyro_sens_a4[2] = 0

         *
         * for testing gyro_sens_a1[0] =  1.; gyro_sens_a1[1] =  0.;
         * gyro_sens_a1[2] =  0.; gyro_sens_a2[0] =  0.; gyro_sens_a2[1] =
         * 1.; gyro_sens_a2[2] =  0.; gyro_sens_a3[0] =  0.; gyro_sens_a3[1]
         * =  0.; gyro_sens_a3[2] =  1.;
         *

        gyro_sens_b1[0] = 0.995782;
        gyro_sens_b1[1] = -0.0917517
        gyro_sens_b1[2] = 0.

        gyro_sens_b2[0] = -0.0648783
        gyro_sens_b2[1] = -0.704124;
        gyro_sens_b2[2] = -0.707107.

        gyro_sens_b3[0] = -0.0648783;
        gyro_sens_b3[1] = -0.704124;
        gyro_sens_b3[2] = 0.707107;

        gyro_sens_b4[0] = -0.5;
        gyro_sens_b4[1] = 0.866025;
        gyro_sens_b4[2] = 0.;

         *
         * for testing gyro_sens_b1[0] =  1.; gyro_sens_b1[1] =  0.;
         * gyro_sens_b1[2] =  0.; gyro_sens_b2[0] =  0.; gyro_sens_b2[1] =
         * 1.; gyro_sens_b2[2] =  0.; gyro_sens_b3[0] =  0.; gyro_sens_b3[1]
         * =  0.; gyro_sens_b3[2] =  1.;
         */

        /* init */
        for (i = 0; i < 8; i++)
                quant_remainder[i]   0.

        /* default gyro a on and gyro b off *
        gyro_onoff_a = 1;
        gyro_onoff_b = 1;
}

void
gyro_onoff(onoff_str
        char            onoff_str[]
{
        if (!strcmp(onoff_str,  irua on"
                gyro_onoff_a = 1;
        if (!strcmp(onoff_str, "irua off )
                gyro_onoff_a = 0;
        if (!strcmp(onoff_str, "irub on" )
                gyro_onoff_b = 1;
        if   strcmp(onoff_str, "irub off
                gyro_onoff_b = 0;


void
gyro_model(rate, gyro_bias)
        d_real          rate[]
        int             gyro_bias[]
{
        int             i;
        d_real          sens_rate[8];
        d_real          gyro_bias_real  gyro_bias_whole;

         * compute rates along gyro sense axes *
        sens_rate[0] = dot3(rate, gyro_sens_a1 ) ;
        sens_rate[1] = dot3(rate, gyro_sens_a2 ) ;
        sens_rate[2] = dot3(rate, gyro_sens_a3 ) ;
        sens_rate[3] = dot3(rate, gyro_sens_a4 ) ;
        sens_rate [41 = dot3 (rate, gyro_sens_b1 ) ;
        sens_rate[5] = dot3(rate, gyro_sens_b2 ) ;
        sens_rate[6] = dot3(rate, gyro_sens_b3 ) ;
        sens_rate[7] = dot3(rate, gyro_sens_b4 ! ;

        /*loss Less quant */
        for (i = 0; i < 9; i++){
                gyro_bias_real = sens_rate[i] / gyro_rate_quant
                        + quant_remainder[i] ;
                quant_remainder[i] . modf(gyro_bias_real , &gyro_bias_whole) ;
                gyro_bias[i] = ( int) gyro_bias_whole;
        }

        /* overwrite with zeros if off */
        if ( !gyro_onoff_a )
                for (i = 0; i < 4; i++)
                        gyro_bias[i] = 0;
        if ( !gyro_onoff_b )
                for (i = 4; i < 8; i++)
                        gyro_bias[i] = 0;

        /' Diagnostics 'f

        if (model .= 4 && diag_time -- > 0) !
                if (diag_flag == 0) !
                        if ( (f_gyro = fopen("/sim/tmp/gyro_diags" , "w") ) ==(FILE *) NUL
L) {
                                printf("File Open Failure for GYRO Diagnostics\n");
                        };
                        diag_flag = 1;

                if (diag_level == 1 !' diag_level == 2 !! diag_level == 3) {
                        if (f_gyro == (FILE *) NULL) {
                                printf("GyroBiases:\n");
                                for (i = 0; i < 8; i++) {
                                        printf("%12.6d\n", gyro_bias[i]);
                                };
                                printf("\n");
                        ! else
                                fprintf ( f_gyro, "GyroBiases:\n"):
                                for (i = 0; i < 8; i++) (
                                        fprintf(f_gyro,"%12 .6d\n", gyro_bias[i] ) ;
                                ;;
                                fprintf ( f_gyro,"\n" ! :
                        };
                };
                if (diag_level    2 ' diag_level == 3)  !
                        if (f_gyro == (FILE *) NULL) {
                                printf("Rates: %12.6f %12.6f %12.6f\n\n", rate[0], rate[
1], rate[2]);
                        ! else {
                                fprintf ( f_gyro,"Rates %12.6f %12 .6f %12.6f\n\n", rate[
0], rate[1], rate[2] ) ;
                        ;;
                };
                if (diag_level == 3) {
                        if (f_gyro == (FILE *) NULL) !
                                for (i = 0; i . 8; i++) !
                                        printf("sens_rate  %12.6f  quant_remainder %12 .6
f \n", sens_rate[i] , quant_remainder[i ! ) ;
                                }
                        ! else {
                                        fprintf ( f_gyro, " sens_rate  %12.6f  quant_remainder
%12.6f \n", sens_rate[i] , quant_remainder[i ! ! ;
                                ;,
                        };
                        if(f_gyro == (FILE *) NULL) !
                                printf("\nEnd of Cycle.  bb_timetag = %f\n\n", bb_timetag);
                        } else {
                                fprintf ( f_gyr0, "\nEnd of Cycle. bb_timetag = %f\n\n" , bb_ti
metag);
                        }
                };

                if (diag_time == 0 && model == 4! !
                        diag_flag = 0;
                        if (f_gyro ' = (FILE *) NULL) /
                                if (fclose(f_gyro) '= 0) !
                                        printf(" Error closing Gyroscope Diagnostic File!\n" )

                                );
                        };
                1;
                if (diag_time < 0 && model == 4)
                        diag_time = 0 ;
);
```

```
/************
 *  r_wheel  *
 ************/

/** DESCRIPTION
$Id: rw.c,v 1.27 1997/01/27 23:02:53 itsl Exp $
This file contains the simulation for the CASSINI reaction wheels for use
in the Integration and Test Laboratory.
**/

/** REVISION HISTORY:
23 June 95:  Jason Bunn  Creation
23 June - 5 July:  Jason Bunn Initial Development
10 July:  Jason Bunn Added Power Model and Integration Coefficients
14 July:  Jason Bunn Began adding interface modules
21 July:  Jason Bunn delivered code to J. Roberts for integration
7 August: Recent Updates :
          'Tachometer Model Corrected
          Bugs fixed to facilitate Integration
          Expanded to four reaction wheels
27 August 96:  Changed to new power model
               changed tachometer function declaration to short
               Changed derivation of scale factor from .1755/128 to .1755 /127
30 August 96:  Coded switch from copies to arrays
20 September 96:  Added code to simulate RTIOU reset
24 September 96:  Added Dahl state friciton limiter and changed scale factor
                  to reflect latest change as per FSC #423, FSW A5.1.0
15 October 96:  Implemented separate current scale factors per FSC #450, FSW A5.2.0
**/

/** INPUTS FOR EACH WHEEL:
Information from the registers concerning the commanded torque.
**/

/** OUTPUTS FOR EACH WHEEL:
Current state of the reaction wheel, including current, tach
output and read back commanded torque.
**/


/** OPERATION:
At the start of a loop, the simulation checks to see if power is on. If, so
the commanded torque is read and added to the state.  If not, the commanded
torque register is not read.  When power is first supplied, the registers are
cleared to zero.  The state is then propagated in time, followed
by an output state where the current state variables of interest are passed
to the registers.  The register file communicates with the RTIOU and passes
the data to the AACS bus through the RTIOU when necessary.  Loops occur
every 62.5  ms.
**/


#include <math.h>
#include <stdio.h>
#include "serverDemo.h"
#ifndef SUN
#include "vme_addresses.h"
#include "assembly.h"

extern int      peek8 ( ) ;
extern int      poke8 ( ) ;
extern char    *assem_rwx1 ;
extern char    *assem_rwx2;
extern char    *assem_rwx3 ;
extern char    *assem_rwx4 ;
extern struct assem_board card[6] ;

#endif
void            RWX_power_on ( ) ;


/** STRUCTURE definitions **/

/* RTIOU Registers */
struct RWA_reg {
        int               power;
        char              reg01wr; /* dummy write to enable torw CMD */
        char              reg04wr; /* load mux control reg */
        char              reg06wr; /* Torque CMD */
        char              reg02rd; /* Dummy read to load tach reg */
        char              reg03rd; /* Upper byte of tach reg */
        char              reg04rd; /* Lower byte of tach reg */
        char              reg05rd; /* A/D Converter */
        char              reg06rd; /* Torque CMD wrap-around */
        char              reg07rd; /* Time out Test */
};

struct RWAParam_s {
        double            cog_amp;
        double            cog_phi ;
        double            dahl_amp;
        double            dahl_ang;
        double            dyn_amp;
        double            dyn_phi ;
        double            int_stp_size;
        double            max_mtr_torq;
        double            phases;
        double            poles;
        double            rate_cutoff;
        double            rate_cuton;
        double            rip_amp;
        double            rip_phi ;
        double            slots;
        double            stat_amp;
        double            stat_phi;
        int               tq_wrd_bts
        int               tach_magnets ;
        double            tach_pos_quant;
        double            visc_amp;
        double            wheel_inertia;
        double            current_scale_factor;
        /**
        cog_amp           -Description:  Reluctance cogging torque amplitude
                           Units :       Nm
        cog_phi           -Description:  Reluctance cogging torque phase
                           Units :       rad
        dahl_amp          -Description:  Coulomb level for Dahl model of bearing
                                         drag torque
                           Units :       Nm
        dahl_ang          -Description:  Amplitude of bearing rotation required for t
he
                                         Dahl model of bearing drag torque to reach 9
8%
                                         of its final value (Coulomb level)
        dyn_amp           -Description:  Amplitude for dynmaic torques due to wheel
                                         products of inertia
                           Units :       Nm/(rad/sec)^2
        dyn_phi           -Description:  Phase angle for dynamic torques due to wheel
                                         products of inertia
                           Units:        rad
int_stp_size    -Description.   Numerical integration step size
                 Units ,        sec
max_mtr_t orq   -Description:   Motor torque limit
                 Units :        Nm
phases          -Description:   Number of motor phases
                 Units :        None
poles           -Description:   Number of motor poles (2*#poles pairs)
                 Units·         None
rate_ cut off   -Description:   RWA rate at which the motor disables motor
                                torque.  A safty feature to prevent RWA damage
                                due to overspeed.
                 Uni ts :       rad/sec
rate_ cut On    -Description:   RWA rate at which the motor re-enables motor
                                torque once it has been shutdown due to an
                                overspeed condition.
                 Un its:        rad/sec
r ip_amp        -Description:   Commutation ripple torque phase
                 Units ,        rad
slots           -Description:   Number of motor winding slots
                 Units :        None
stat_ amp       -Description:   Amplitude for static forces due to wheel
                                center of mass offset from spin axis
                 Units :        N/(rad/sec)^2
stat_ph i       -Description:   Phase angle for static forces due to wheel
                                center of mass offset from spin axis
                 Units:         rad
tach_magne ts   The number of magnets in the tachometer. (At least 2
                are required)  A double variable. Unitless

t q_wr d_b t S  -Description:   Motor torque word lenght
                 Units ,        bits
vi sc_amp       -Rescript ion.: Viscous fritction torque for bearing
                 Un its:        Nm/(rad/sec)
•.,

/* derivied */
double          cog_freq;
double          dahl_sig;
double          rip_ freq;
double          torq_const;
double          w_cog_cutoff ;
double          w_rip_cutof f;
doubl e         w_sd_cut off ;

/**
cog_freq        -Description:   Reluctance cogging torque" frequency"
                 Unit ,         cycles/rad
dahl_sig        -Description:   Dhal rest slope (change in friction torque
                                per change in bearing angle at torque=0
                                point) for Dahl model of bearing drag torque
                 Uni ts :       Nm/rad
rip_freq        -Description:   Commutation ripple torque " frequency"
                 Units ,        cycles /rad
torq_const      -Description:   Mot or torque resolution
                 Units :        Nm per bit
w_cog_cutoff    -Description:   Angular rate at which the reluctance cogging
                                torque frequency reaches the Nyquist sampling
                                frequency.  At this point, cogging torques
                                would be ali ased in the simulation, so they
                                are set to zero
                 Units :        rad/sec
w_rip_cutoff    -Description:   Angular rate at which the commutation ripple
                                torque frequency reaches the Nyquist sampling
                                frequency.  At this point, cogging torques
                                would be aliased in the simulation, so they
                                are set to zero.
                 Units·         rad/sec
        w—sd—cutoff     -Description:   Angular rate at which the commutation ripple
                                        torque frequency reaches the Nyquist samplin
9
                                        frequency.  At this point , cogging torques
                                        would be aliased in the simulation, so they
                                        are set to zero
                         Units:         rad/ sec
        **I

};

/* State Vector and State Derivative */

struct wheel_ state!
        double          position;       /' Position of Wheel */
        double          rate;    /* Angular Velocity of Wheel● /
        double          dahl;    /* dahl friction term */
};

struct wheel_ deriv {
        double          posd;    /* Position Derivative (Angular Velocity) */
        double          rated;   /* Angular Velocity Deviv. (Angular
                                    Acceleration) '1
        double          dahld;   /* Dahl friction term rate of change '/
};

struct RWAParam_s rwa_model'4 ] ;


struct RWA_r e g  rwa_reg!4 ];


struct wheel_state state[4];


struct wheel_state temp_state_1[4],


struct wheel_ state temp_state_2(4];


struct wheel_state temp_state_3[4];


struct wheel_deriv stated1[4];
struct wheel _deriv stated2[4];
struct wh eel_der iv stated3[4];
struct wheel _deriv stated4[4];

/** Global Variables ** /

double          dt = 0.0625;

double          coef_1 = 0.5;
double          coef_2 = 0.5;
double          coef_3 = 1;
double          coef_4 = 0.166667;
double          coef_5 = 0.333333;
double          coef_6 = 0.233333;
double          coef_7 = 0.166667;

double          fractional _prev_tach [4] ;
```

```
doubl e         overspeed [4] ;
double          nxt_overspeed [ 47 ;
double          power [4 ! ;
double          torq_word[4] ;
double          torque [4] ;
short           tach [4];
short           old_tach_output [4] ;
short           tach_output [4] ;
int             rw_init [4] ;
char           ●assem_rwx[4!  ;

int             rwa_flag[4] = {0, 0, 0, 0};



I "' FUNCT IONS **/
#i fdef SUN
char
peek8 (char 'x)

        return 'x;


char
poke8(char 'x, char c)

        *x = c;
        return c;

#endif

/* dsign */
double
dsign(
      double a,
      double b)

        if (b >= 0)
                return fabs((float) a) :
        return -fabs((float) a) ;
;;


/* Initial Parameters */
void
R WA_load_def (rwp, i )
        str uct RWAParam_s *rwp ;
        int             i;
(
        FILE           ●fp;
        char           *a;
        if (i = 0)
                a . " /sim/dynami cs/rw_param s_1 " ;
        if (i = 1)
                a = " /sim/dynam ics/rw_params_2" ;
        if (i = 2)
                a = " /sim/dynam ics/rw_params_3 " ;
        if (i . 3)
                a = " /sim/dynami cs/rw_params_4" ;

        if ((fp = fopen (a, "r" )) '= NULL) {
                fscanf(fp, "%lf", &rwp->cog_amp) ;
                fscanf(fp,"%lf", &rwp->c og_phi) ;
                fscanf(fp,"%lf", &rwp->dah l_amp) ;
                fscanf(fp, "%lf ", &rwp->dahl_ang) ;
                fscanf (fp, "%lf" , &rwp->dyn_amp) ;
                fscanf(fp,"%lf" , &rwp->dyn_phi) ;
                fscanf ( fp,  "%lf",&rwp->in t_s tp_size) ;
                fscanf(fp,  "%lf",  &rwp->max_mtr_to rq) ;
                fscanf (fp,"%lf" , &rwp->phases) ;
                fscanf (fp,"%lf" , &rwp->poles) ;
                fscanf(fp,  "%lf", &rwp->rate_cutoff) ;
                fscanf(fp,  "%lf", &rwp->rate_cut on) ;
                fscanf (fp,"%lf" , &rwp->rip_amp) ;
                fscanf(fp,"%lf" , &rwp->rip_phi) ;
                fscanf(fp,"%lf" , &rwp->sl ots) ;
                fscanf(fp,"%lf" , &rwp->stat_amp) ;
                fscanf(fp,"%lf" , &rwp->stat_phi) ;
                fscanf( fp,"%d" ,  &rwp->tq_wrd_bts) ;
                fscanf ( fp, "%lf" , &rwp->visc_amp);
                fscanf(fp,"%d" ,  &rwp->tach_magne ts) ;
                fscanf(fp,  "%lf", &rwp->wheel_iner tia) ;
                fscanf(fp,  "%lf", &rwp->current_sca le_fact or) ;
                fclose(fp) ;
        };

};


void
RWX_power_off ( strut: RWA_reg•rwp, char *ass em_base )

        /' Set commanded torque to zero and power to off•/
        rwp->reg 06wr = 0;
        rwp->power . 0;
        /* Clear RAM so that power on functions smoothly '/
        poke  8(assem_base + 2 *1+1,0);
        poke8(assem_base + 2 • 2 + 1,0) ;
        poke8(assem_base + 2 * 3 + 1,0) ;
        poke8(assem_base + 2 ' 4+1,0) ;
        poke8 (assem_base + 2 ' 5 + 1,0);
        poke8(assem_base + 2 ' 6+1,0) ;
        poke8(assem_base + 2 * 7 +1,0) ;
.,

/* Read information from registers */
void
RWX_getbyte(struct RWA_reg * rwp, char *assem_base, short *tach)

        int             x:
        /**
                x= peek 8 (assem_base  .  2 . Poweronregister+1);
        ../
        x . read_ rwa_power (assem_base) ;
        if (x == 1) {           /* Is power on? */
                if (rwp->power == 0) {  /* If power is on, is it a POR ? */
                        RWX_power_on ( rwp, Lath) ;
                };
                / * get Torque Enable Command ' /
                rwp->reg 01wr = peek8 (assem_base + 2 * 1 + 1) ;
                I' get 2 ' S complement Torque Command * I
                rwp->reg06wr = peek8 (assem_base + 2 ' 5 + 1 ) ;
        } else {
                RWX_powe r_of f ( rwp, assem_base ) ;
        };
};

/* h'rice information to registers */
void
RWX_putbyte ( struct RWA_reg•rwp, char ●asse.m_base )
```

```
        /** echo Torque command back to FSW
        poke8(assem_base + 2 ' 6 + 1, rwp->reg06rd) ;
 Asem sim card uses a RAM, therefore, we do not want to overwrite new
torque  information. **/

        /* A/D Converter info */
        poke8(assem_base + 2 ' 5 + 1, rwp->reg05rd) ;

        /* Upper Byte of Tachometer Output */
        poke8(assem_base + 2 ' 3 + 1, rwp->reg03rd) ;

        /* Lower Byte of Tachometer Output */
        poke8(assem_base + 2 * 4 + 1, rwp->reg 04rd) ;

        /*Dummy read to load tach register */
        poke8(assem_base + 2 * 2 + 1, rwp->reg02rd) ;

        /* Time out Test */
        poke8(assem_base + 2 * 7 + 1, rwp->reg07rd);
};


1' Dahl Friction */
/* Adpopted from Matt Wette's Code */
/*
 * This routine models the analog portion of the Cassini Reaction Wheel
 * Assembly (RWA) Dahl friction model.
 *
 * The routine accepts the current value of the RWA bearing angle, bearing
 * velocity, and the current value of the Dahl model bearing and friction
 * torque.  It then returns the time derivative of the Dahl model bearing
 * friction torque.
 ●-
double
RWA_dahl ( struct RWAParam_s•rwp,
         struct wheel _state * state,    /' gimbal angle from dyn (rad) */
/* angle rate  from dyn (rad/see) '/
/* Dahl fric from dahl-model (Nm) * /

         double fric_do t)
{                                /* time deriv of dahl fric (Nm/s) "1
        double          dahl_fric_lim;
        double          dtorq_dangl e;
        double          t emp;

        /* Compute the time derivative of the Dahl bearing torque : */
        if (fabs(state->dahl) >= (rwp->dahl_amp) ) (
                dahl_fric_lim = dsign( (rwp->dahl_amp) , state ->dahl) ;
        } else {
                dahl_fric_lim . state ->dahl ;
        )

        temp = 1.0 + dsign(1.0, state ->rate) * dahl_fric_lim / (rwp->dahl_amp) ;
        dtorq_dangle . -(rwp->dahl_sig) ' dsign(1 .0, temp) ' fabs(temp) ;
        (fric_dot) = state ->rate , dtorq_dangle;
        return ( fric_dot) ;
};


/* Power Model  Current = Power I 30 Volts */
void
rwa_pwr (
        double rw_rate,
        double mtr_tq,
                double ●pto Eal)
/**
@doc@  @name rwa_pwr
@doc@  @disc This subroutine calculates the power usage for a single reaction
@doc@  @disc wheel based on wheel speed and applied torque values
@doc@  @returns Ptotal
@doc@  @req model the analog portion of the reaction wheel
●*,


/**
 •Inputs :  rw_rate - wheel rate (rad/see) mtr_tq - torque from motor (Nm)
 *
 * Outputs: Ptotal - Total power usage (Watt s)
 ●*,

/ * Based on Appenix to CAB dated 11-94 changed on 5-24-96 by L .Montanez */

/** New model based on Model Validation Team input 8/9/96
 •Changed  9/27/96  by J. Bunn
 .*,


        double          c0Power, c1Power, c2Power, c3 Power, c4Power;
        c0Power . 9.4999355;    /* Watts */
        c1 Power = 60.4512337;  /* Watts/ (N-m)● I
        c2Power . 1,03'54894;   /* Watts/ (N-m) per rad/sec● I
        c3Power = 0.00280 60;   /* Watts/ (rad/sec */
        c4Power = 4. 00;        /* Watts */

        if (mtr_tq < 0.0)
                *ptotal = (c0Power + (c1Power * fabs(mtr_tq)) -
                        (c3Power + c2Power * fabs(mtr_tq)) * rw_rate);
        else
                *ptotal = (c0Power + (c1Power * mtr_tq) +
                        (c3Power + c2Power * mtr_tq) * rw_rate);
        if (*ptotal < 0.0)
                *ptotal = c4Power;
        else
                *ptotal +. c4 Power;
};

void
RWA_ana(
        struct RWAParam_s , rwp,
        double torq_word,
        struct whe el_state ' state,
        double *ovr_spd_f lg,
        double *torq,
        double *power ,
        double * nx t_ o_s_f )


        /**
        @doc@  @name R WA_ana
        @doc@  @disc This routine accepts the current value of the RWA torque cmd,
        @doc@  @disc bearing angle, bearing velocity, and the current vlaue of the
        @doc@  @disc Dahl model bearing friction torque and overspeed flag
        @doc@  @returns torq, stat _force1, stat _force2, dyn_tor que1, dyn_torque2, power
        @doc@  @returns nxt_o_s_f
        @doc@  @functions_called fabs, rwa_pwr
        @doc@  @req model the analog portion of the reaction wheel
        **/

        double          cogging_t orq;
        double          motor_torq;
        double          ripple_torq;
        double          visc_fric;
        double          x;
        /*
         * Check for an RWA overspeed condition and compute the next value of
         * the overspeed indicator flag.  A non-zero value for the flag
         * indicates an overspeed condition.
         */

        if  *ovr_spd_flg == 0.0) {
                if (fabs(state->rate) >= (rwp->rate_cutoff)) {
                        *nxt_o_s_f = 1.0;
                  else {
                        *nxt_o_s_f = 0.0;

        } else {
                if (fabs(state->rate) <= (rwp->rate_cuton)) {
                        *nxt_o_s_f = 0.0;
                } else {
                        *nxt_o_s_f = 1.0;
                }


        *ovr_spd_flg = *nx=_o_s_f;

         *
         * Compute motor torque   If an overspeed condition exists, null the
         * motor torque:
         */
        if  *ovr_spd_flg == 0.0) :
                motor_torq = torq_word *  rwp->torq_const ,
        } else
                motor_torq = 0.0;
        }

        /* Compute commutation ripple torque: */
        if (fabs(state->rate) >= (rwp->w_rip_cutoff)) {
                ripple_torq = 0.0;
        } else {
                x = (rwp->rip_freq) * state->position + (rwp->rip_phi);
                ripple_torq = (rwp->rip_amp) * motor_torq * sin(x);



        /* Compute viscous friction torque: */
        visc_fric = -(rwp->visc_amp) * state->rate;

         *
         * Compute the total torque that will be applied along the RWA gimbal
         * in the dynamics simulation:
         */
         *torq) = motor_torq + ripple_torq + state->dahl + visc_fric;

        /* Compute the power used by the RWA: */
        rwa_pwr(state->rate, motor_torq, power);
};

/* Calculate State Derivative */
void
rw3_deriv(
          struct wheel_state * state,
          double torq_word,
          struct wheel_deriv * stated,
          double *over_speed,
```

```
             double *nxt_overspeed,
             struct RWAParam_s * params
             double *torque,
             double *power)
{
        RWA_ana(params, torq_word, state, over_speed, torque, power, nxt_overspeed).

        stated->posd = state->rate;
        stated->rated = *torque / params->wheel_inertia;
        stated->dahld = RWA_dahl(params, state, stated->dahld),


/* Load Inital Values */
void
RWX_load_def(
               struct RWA_reg * rwp,
               short *tach_reg)
{                                       /* rwa structure */
        rwp->power = 0;
        rwp->reg06wr = 0,
        rwp->reg02rd = 0,
        rwp->reg03rd = 0,
        rwp->reg04rd = 0,
        rwp->reg05rd = 0,
        rwp->reg06rd = 0,
        *tach_reg = 0=
}

/* Power On Procedures */
void
RWX_power_on(
               struct RWA_reg   rwp, short *tach)
                                        /* rwa structure */
        rwp->power = 1;
        rwp->reg02rd = 0,
        rwp->reg03rd = 0,
        rwp->reg04rd = 0,
        rwp->reg05rd = 0,
        rwp->reg06rd = 0,
        *tach = 0;
}

/* Initialzation Routine */
void
RWA_upd_params(struct RWAParam_s * rwp)

        rwp->cog_freq = (rwp->slots);
        rwp->dahl_sig = 4.0 * (rwp->dahl_amp) / (rwp->dahl_ang);
        rwp->rip_freq = (rwp->phases) * (rwp->poles);
        rwp->torq_const = (rwp->max_mtr_torq) /
              ((1 << (rwp->tq_wrd_bts - 1)) - 1);
        rwp->w_cog_cutoff = 3.14159 / ((rwp->cog_freq)
                                      * (rwp->int_stp_size));
        rwp->w_rip_cutoff = 3.14159 / ((rwp->rip_freq)
                                      * (rwp->int_stp_size));
        rwp->w_sd_cutoff = 3.14159 / (rwp->int_stp_size);
        rwp->tach_pos_quant = 2.0 * 3.14159265358979 / rwp->tach_magnets;
};

/* Initialize State and State Derivitive Structures */
void
state_init(struct wheel_state * state)

        state->position = 0.0;
        state->rate = 0.0;
        state ->dahl = 0 0;


void
stated_ini t(s truct wheel _deriv ' stated)
(
        stated->posd = 0 0;
        stated-> >rated = 0 0;
        stated- >dahld = 0 0;
}
#ifdef OLD
double
tachometer (struct RWAParam_s * rwa_model, struct wheel _state * state, short *old_ tach_ou
tput, short *tach_ output )
{
        double          tach_ quan t;
        double          inc_tach_output;

        *old_ tach_output . * tach_output ;
        tach_quant = 2 0 ' 3 14 1592654 / rwa_model ->tach_magnets;
        inc_tach_outpu t = (stat e->rate / tach_quant) - *o ld_tach_ou tput;
        *tach_output . ● old_cach_ou:p,Jt . inc_tach_output ;
        return ( inc_tach_output ) ;
};
#endi f

short
tachometer(struct RWAParam_s * rwa_model, struct wheel_state * state, double *fractional
_prev_tach)
{
        double          real_tach;
        double          ideal_tach;
        double          tach_quant;

        tach_quant . 2 0 ' 3.141592654 / rwa_model ->tach_magnets;

        ideal _tach . (sta te->ra te / tach_quant) * .125;
        real _tach = ideal _tach . *fracti onal_prev_ta ch;
        *fracti onal _prev_tach = real_tach - ( int) real _tach;
        return ( (short) real _tach) ;
}

void
Output(struct RWA_reg * rwa_reg, struct RWAParam_s * rwa_model, struct wheel_state * sta
te, double *power, short *tach, double *fractional_prev_tach)
{
        short           temp_tach;
        short           overt ach;
        short           under tach;
        double          current;

        current = ( (*power / 30.0) / (rwa_model ->current_sca le_fac tor) ) + 128.0;
        rwa_reg->reg05rd . current;
        rwa_reg->reg C 6rd . rwa_reg->reg 0 6wr;
        ● tach . tachome ter(rwa_model , state, fracti onal _prev_tach) ;
        temp_tach . *tach;
        overtach = 0;
        undertach = 0;
        if (*tach > 2047)
                overtach = 0x8000;
        if (*tach < -2048)
                undertach = 0x4000;
        temp_tach = ( *tach & 0xFFF) | overtach undertach;
        rwa_reg->reg03rd . (temp_tach >> 8 & 0xFF) ;
        rwa_reg->reg04rd = (temp_tach & 0xFF) ;
```

```
        rwa_reg->reg02rd = (0 xFFFF & 0xFF) ;
        rwa_reg->reg 07rd = (0 xFF) ;
};

#ifdef SUN
char            assem_rwx1[0xfff f] ;
char            assem_rwx2 [0xffff] ;
char            assem_rwx3 [0xffff] ;
char            assem_rwx4[0xff ff] ;
#endi f

int
read_ rwa_power ( base)
        char            *base;
{
        int             power, pwr_status;

        pwr_status . peek8 (base + (0x8000 + 1) ) ;
        power = 0;
        if ( (pwr_status & 0x04) == 0x04)
                power . 1;

        return (power! ;
};


int
rwa_init ( )
{
        int             i;


        for (i = 0; i < 4; i++)(
                /* Initialize */
                fraction al _prev_tach [i] = 0. 0;
                overspeed[i] = 0;
                nxt_overspeed [ i ] = 0;
                power[i] . 0;
                torq_word[i] . 0;
                torque [i] . 0;
                tach [i] . 0;
                old_tach_output [ i] . C ;
                tach_output[i] = 0;
                rw_init [i] = 0;


                /*Reset registers tozero and clear  any  accumulators */
                RWA_load_def (&rwa_model ! i] , i ) ;
                R WA_upd_params ( &rwa_model [ i ] ) ;
                RWX_load_def (&rwa_reg ! i ] , &tach [i ] ) ;

                /* Initialize state and derivitive structures● I
                state_ init(&state[i] ) ;
                state_ ini t(&temp_state_1 [i] ) ;
                state_ init(&temp_state_2 [i] ) ;
                state_ init(&temp_s tate_3 [i] ) ;

                stated_ ini t(&stated1 [i! ) :
                stated_init (&stated2 [i] ) ;
                stated_ init(&stated3 [i] ) ;
                stated_ini t(&stated4 [i] ) ;

                rw_init [i) = 1;

        assem_rwx[0] = assem_rwx1;
        assem_rwx[1] = assem_rwx2;
        assem_rwx[2] = assem_rwx3;
        assem_rwx[3] = assem_rwx4;
        return 0;
};


int
rwa_loop()
{
        int             i;
        for (i = 0; i < 4; i++){
                /* Read Registers */
                RWX_getbyte(&rwa_reg[i], assem_rwx[i], &tach[i]);
                if(rw_init[i] == 0)
                        return 0;

                if((rwa_reg[i].reg01wr != 0) && (rwa_flag[i] == 1))
                        rwa_flag[i] = 2;
                else if ((rwa_reg[i].reg01wr != 0) && (rwa_flag[i] == 2))
                        rwa_flag[i] = 0;
                else if ((rwa_reg[i].reg01wr != 0) && (rwa_flag[i] == 0)) {
                        poke8(assem_rwx[i] + 2 * 6 + 1, 0);
                        rwa_reg[i].reg06wr = 0;
                };
                if (rwa_reg[i].reg01wr == 0 && rwa_reg[i].power == 1){
                        rwa_flag[i] = 1;
                        poke8(assem_rwx[i] + 2 * 1 + 1, 0xFF);
                };

                /* Compute Torque Word */
                if (rwa_reg[i].reg06wr > rwa_model[i].max_mtr_torq / rwa_model[i].torq_c
onst)
                        torq_word[i] = dsign(rwa_model[i].max_mtr_torq / rwa_model[i].to
rq_const, rwa_reg[i].reg06wr);
                else
                        torq_word[i] = rwa_reg[i].reg06wr;



                /* Propagate State */
                /*
                 * 4th order numerical integration. Time step = dt = 0.0625
                 * seconds. Caculates the next values of the state variables
                 */

                rwa_deriv(&state[i], torq_word[i], &stated1[i], &overspeed[i], &nxt_over
speed[i],
                          &rwa_model[i],&torque[i],&power[i]);
                temp_state_1[i].position = state[i].position + coef_1 * dt * stated1[i].
posd;
                temp_state_1[i].rate = state[i].rate + coef_1 * dt * stated1[i].rated;
                temp_state_1[i].dahl = state[i].dahl + coef_1 * dt * stated1[i].dahld;

                rwa_deriv(&temp_state_1[i], torq_word[i], &stated2[i], &overspeed[i], &n
xt_overspeed[i],
                          &rwa_model[i], &torque[i], &power[i]);
                temp_state_2[i].position = state[i].position + coef_2 * dt * stated2[i].
posd;
                temp_state_2[i].rate = state[i].rate + coef_2 * dt * stated2[i].rated;
                temp_state_2[i].dahl = state[i].dahl + coef_2 * dt * stated2[i].dahld;

                rwa_deriv(&temp_state_2[i], torq_word[i], &stated3[i], &overspeed[i], &n
xt_overspeed[i],
                          &rwa_model[i], &torque[i], &power[i]);
```

```
                temp_state_3[i].position = state[i].position + coef_3 * dt * stated3
[i].posd;
                temp_state_3[i].rate = state[i].rate + coef_3 * dt * stated3[i].rate
d;
                temp_state_3[i].dahl = state[i].dahl + coef_3 * dt * stated3[i].dahl
d;

                rwa_deriv(&temp_state_3[i], torq_word[i], &stated4[i], &overspeed[i]
, &nxt_overspeed[i],
                          &rwa_model[i], &torque[i], &power[i]);


                state[i].position += dt * (coef_4 * stated1[i].posd + coef_5 * state
d2[i].posd +
                          coef_6 * stated3[i].posd + coef_7 * stated4[i].posd);
                state[i].rate += dt * (coef_4 * stated1[i].rated + coef_5 * stated2[
i].rated +
                          coef_6 * stated3[i].rated + coef_7 * stated4[i].rated);
                state[i].dahl += dt * (coef_4 * stated1[i].dahld + coef_5 * stated2[
i].dahld +
                          coef_6 * stated3[i].dahld + coef_7 * stated4[i].dahld);

                /* Dahl Model Friction Limiter */

                if (fabs(state[i].dahl) >= (rwa_model[i].dahl_amp))
                        state[i].dahl = dsign(rwa_model[i].dahl_amp,state[i].dahl);

                /* Output Registers */

                Output(&rwa_reg[i], &rwa_model[i], &state[i], &power[i], &tach[i], &
fractional_prev_tach[i]);

                if (card[i+2].assem_write_ok != 0) {
                  RWX_putbyte(&rwa_reg[i], assem_rwx[i]);
                  card[i+2].assem_write_done = -1;
                };
        }
        return 0;
```

```
/* rw_model c '/
/* cassini itl reaction wheel model '/

/* for the d_real typedef */
#include "types-darts h"
/* for PI */
#include "generic- darts .h"

/* standard */
#include <string. h>

/* local "/
#include " rw_model h"

/* Diagnostics */
extern int      model ;
extern int      diag_time;
extern int      diag_level ;
extern FILE    ' f_rwa;
extern int      diag_flag;
extern d_real   bb_timetag;

/* namelist data */
d_real          rw_num_magnets, rw_ra t e_es t_Wn, rw_c ont_Wn, rw_whee l Inert i a, rw_v i sc_amp

                rw_dt, max_mtr_trq, trq_wrd_bits;

/* derived parameters (init) */
d_real          rw_tach_scale_factor, rw_lowPassGain2, rw_lowPassGain1, rw_kp,
                rw_ki, trq_scale_factor;,

/* local to model */
d_real          tach_t_tag_save[4], rw_rate_est[4], rw_drag_tq_est[4];
d_real          old_trq_com[4];
int             rwa_onoff[4], old_tach_out[4], last_pwr[4];

/** CHANGE LOG: **/
/** Modification 29 July 96 by J. Bunn:
Changed onoff to rwa_onoff to avoid conflict with other models
Changed wheel interia to 0.16146 to match rw_params files
Changed number of magnets to 24 to match rw_params files
Added scale factor to convert from dn to torque in rw_model.c
**/

/** Modification 21 August by R. Okuno, J. Bunn, D. Garcia:
Added filter to limit tach count to max that can be expected (100 per RTI)
Added filter to limit torque command to max (0.1755)
Added filter to limit maximum drag torque
●*,

/** Modification 23 August by J. Bunn, ?.. Okuno :
Added filter to limit tim etag to between 1 and .15 seconds
●+,

I ** Recent Enhancements :
1.  Converted scale factor to variable calculated in rw-init (8 /27/96 J. Bunn)
2. Updated rw_visc_amp to match CAB (9/27/96 J. Bunn)
3.  Added code to set rate estimate to rate sample if wheels are just being turned on ( P
/27/96 J. Bunn)
●.,

/**  11/22/95- Updated to have latest max motor torque
**,
void
rw_init( )
```

```
        int             i;

        /* namelist parameters */
        rw_dt = 0.125;
        rw_wheel Inertia = 0.15146;
        rw_num_magnets = 24. ;
        rw_visc_amp = 1.14 6e-4;
        rw_rate_est_Wn = 0 1 ;    /* in Hz */
        rw_cont_Wn = 0 01;        /* in Hz */
        max_mtr_trq = 0. 17399;
        trq_wrd_bits = 8;

        /* derived parameters '/
        trq_scale_fact or . max_mtr_trq / (1 << ( (unsigned int) trq_wrd_bits - 1) - 1
);

        rw_tach_sca le_factor = 2 " PI I rw_num_magnets;
        rw_lowPassGain2 = exp(- (2.  * PI•rw_rate_est_Wn) * rw_dt ) ;
        rw_lowPass Gain1 = 1 - rw_lowPass Gain2 ;
        rw_kp = 2. ' 0.707 •(2 * PI * rw_cont_Wn) * rw_wheel Inertia;
        rw_ki = pow( (2 * PI•rw_cont_Wn) , 2 ) •rw_wheel Inertia;

        /* initialize '/
        for (i = 0; i < 4; i++){
                tach_ t_tag_save ! i! = 0 ;
                rw_rate_est [i ! = 0 ;
                rw_drag_tq_est ' i 1 = 0 ;
                old_tach_out [i] = 0;
                old_trq_com[i] - 0;
                last_pwr[i] = C;


        /* default wheels on */
        for (i = 0; i < 4; i++)
                rwa_onoff[i] = 1;


void
rw_onoff(onoff_str)
        char            onoff_str[];

        if ( !strcmp(onoff_str, "rw1 on"))
                rwa_onoff ! 0 ! = 1;
        if ( !strcmp(onoff_str, "rw1 off"))
                rwa_onoff [ 0] = 0;
        if ( !strcmp (onoff_str, "rw2 on"))
                rwa_on off ! 1 ! = 1;
        if ( !strcmp(onoff_str, "rw2 off"))
                rwa_onoff [1 ! . 0;
        if('!strcmp(on of f_s tr, "rw3 on"))
                rwa_onof f ! 2 ] = 1;
        if ( strcmp(onoff_str, "rw3 off"))
                rwa_on off [2] = 0;
        if ( !strcmp(onoff_str, "rw4 on"))
                rwa_onof f ! 3 ! = 1;
        if''strcmp(onoff_s tr, "rw4 off"))
                rwa_onof f ! 3 ] = 0;


void
rw_model ( temp_trq_com, trq_app  new_tach_out, tach_t_tag, rates, pwr_st)
        d_real          temp_trq_com[], trq_app[], tach_t_tag[], rates[];

        int             pwr_st ! ];
        int             new_tach_out [] ;
```

```
 *
 * in:  temp_trq_com      commanded wheel torques tach_out          delta
 * pulses tachs tach_t_tag  tach read time tags rates             wheel rates
 * from DARTS pwr_st            wheel power states
 *
 * out  trq_app      torque applied to DARTS wheels
 */

{
        d_real          new_trq_com[4];
        d_real          diff_time, rate_error, corr_tq, rate_sample;
        int             i, tach_out[4];
        d_real          trq_com[4];


        /** Limiter to bound incoming trq command to +- .1755 **/

        for  i = 0; i < 4; i++) {
                ** Commanded torque comes in as dn from Bus Monitor.  Needs to be
                * changed to torque.  This is done via the trq_scale_factor
                **/

                new_trq_com[i] = temp_trq_com[i] * trq_scale_factor;
                if (new_trq_com[i] >= 0.0) {
                        if (new_trq_com[i] > .1755)
                                trq_com[i] = old_trq_com[i];
                        else
                                trq_com[i] = new_trq_com[i];
                } else {
                        if (new_trq_com[i] < -.1755)
                                trq_com[i] = old_trq_com[i];
                        else
                                trq_com[i] = new_trq_com[i];
                }

                old_trq_com[i] = trq_com[i];

        /* Limiter to bound incoming tach data to between -100 and 100 */
        for (i = 0; i < 4; i++) {
                if (new_tach_out[i] >= 0) {
                        if (new_tach_out[i] > 100)
                                tach_out[i] = old_tach_out[i]
                        else
                                tach_out[i] = new_tach_out[i];
                } else
                        if (new_tach_out[i] < -100)
                                tach_out[i] = old_tach_out[i];
                        else
                                tach_out[i] = new_tach_out[i];
                }
                old_tach_out[i  = tach_out[i];


        for (i = 0; i < 4; i++) {

                /* if off just spin down */
                if ((!rwa_onoff[i]) || (!pwr_st[i])  {
                        trq_app[i] = -rw_visc_amp * rates[i]
                        last_pwr[i] = 0;
                } else {
                        diff_time = tach_t_tag[i] - tach_t_tag_save[i].
                        tach_t_tag_save[i] = tach_t_tag[i];
                        if ((diff_time == 0.0) || (diff_time > 0.250))
                                rate_error = 0.;
                        } else {
```

```
                                if (diff_time < 0.10 || diff_time > 0.15)
                                        diff_time = 0.125;
                                /** new tach read performed - compute new rate estim
ate **
                                rate_sample = tach_out[i] * rw_tach_scale_factor / d
iff_time;

                                /* rate estimator */
                                if (last_pwr[i] == 0) {
                                        rw_rate_est[i] = rate_sample;
                                        last_pwr[i] = 1;
                                } else
                                        rw_rate_est[i] = rw_lowPassGain1 * rate_samp
le +
                                                rw_lowPassGain2 * rw_rate_est[i];
                                rate_error = rw_rate_est[i] - rates[i];
                        };

                        /* controller torque correction computation */
                        corr_tq = rw_drag_tq_est[i] - rw_kp * rate_error;
                        rw_drag_tq_est[i] -= rw_ki * diff_time * rate_error;
                        /* Limiter = 1.5(dahl_amp+visc_amp*max_speed) */
                        if (rw_drag_tq_est[i] > (1.5 * (3.22e-4 + 1.146e-4 * 210)))
                                rw_drag_tq_est[i] = (1.5 * (3.22e-4 + 1.146e-4 * 210
  ;
                        if (rw_drag_tq_est[i] < -(1.5 * (3.22e-4 + 1.146e-4 * 210)))
                                rw_drag_tq_est[i] = -(1.5 * (3.22e-4 + 1.146e-4 * 21
0));

                        trq_app[i] = trq_com[i]   corr_tq;

                }

                /* Diagnostics */

                if (model == 6 && diag_time > 0) {
                        if (diag_flag == 0) {
                                if ((f_rwa = fopen("/sim/tmp/rwa_diags", "w")) == (F
ILE *) NULL)
                                        printf("File Open Failure for RWA diagnostic
s

                                }
                                diag_flag = 1;
                        };
                        if (diag_level == 1 || diag_level == 2 || diag_level == 3) {
                                if (f_rwa == (FILE *) NULL)
                                        printf("RWA  %d Outputs: Torque: %5.6f \n",
i, trq_app[i])
                                } else {
                                        fprintf(f_rwa, "RWA  %d Outputs: Torque: %5
6f \n", i, trq_app[i

                        };
                        if (diag_level == 2 || diag_level == 3) {
                                if (f_rwa == (FILE *) NULL) {
                                        printf("       Inputs: temp_trq %5.6f  temp_
t_tag %12.6f  new_tach %5d  rates %5.6f  power %d\n"  temp_trq_com[i]  tach_t_tag[i]
, new_tach_out[i], rates[i], pwr_st[i]);
                                } else {
                                        fprintf(f_rwa, "       Inputs: temp_trq %5.6
f  temp_t_tag %12.6f  new_tach %5d  rates %5.6f  power %d\n", temp_trq_com[i], tach_
t_tag[i], new_tach_out[i], rates[i], pwr_st[i]);
                                }
                        };
                        if  diag level == 3) {
                                if (f_rwa == (FILE  *) NULL) {
                                        printf."      Internals  t_tag_save  %12.6f
  rate_est %5.6f  drag_tw %2.6f  old_tach %5d  old_torq %3.6f\n ", tach_t_tag_save[i], r
w_rate_est[i], rw_drag_tq_est[i], old_tach_out[i], old_trq_com[i]);
                                        printf("                    last_pwr %d  onoff %d
new_trq %3.6f  tach_out %5d  trq_com %3.6f\n\n", last_pwr[i], rwa_onoff[i], new_trq_com[
i], tach_out[i], trq_com[i]);
                                } else {
                                        fprintf(f_rwa, "       Internals: t_tag_save  %1
2.6f  rate_est %5.6f  drag_tw %2.6f  old_tach %5d  old_torq %3.6f\n ", tach_t_tag_save[i
], rw_rate_est[i], rw_drag_tq_est[i], old_tach_out[i], old_trq_com[i]);
                                        fprintf(f_rwa, "                          last_pwr %d  on
off %d  new_trq %3.6f  tach_out %5d  trq_com %3.6f\n\n", last_pwr[i], rwa_onoff[i], new_
trq_com[i], tach_out[i], trq_com[i]);
                                };




        if (model == 6) {
                if (diag_time > 0) {
                        if (f_rwa == (FILE *) NULL) {
                                printf("End of Cycle.  bb_timetag =  %f  \n\n\n\n", bb_t
imetag);
                                diag_time--;
                        } else {
                                fprintf(f_rwa, "End of Cycle.  bb_timetag =  %f  \n\n\n\
n", bb_timetag);
                                diag_time--;
                        };
                } else if (diag_time == 0) {
                        diag_flag = 0;
                        diag_time = -1;
                        if (f_rwa != (FILE *) NULL) {
                                if (fclose(f_rwa) != 0) {
                                        printf("Error closing RWA Diagnostic file!\n");


                } else if (diag_time <= 0 && model == 6) {
                        diag_time = -1;
```